
Brian M. Adams, Mohamed S. Ebeida, Michael S. Eldred, John D. Jakeman, Laura P. Swiler, J. Adam Stephens, Dena M. Vigil, Timothy M. Wildey
Optimization and Uncertainty Quantification Department

William J. Bohnhoff
Radiation Transport Department

Keith R. Dalbey
Mission Analysis and Simulation Department

John P. Eddy
System Readiness and Sustainment Technologies Department

Russell W. Hooper
Multiphysics Applications Department

Kenneth T. Hu
Validation and Uncertainty Quantification Department

Lara E. Bauman, Patricia D. Hough
Quantitative Modeling and Analysis Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185

Ahmad Rushdi
Institute for Computational and Engineering Sciences
The University of Texas at Austin
P.O. Box 4.102
Austin, TX 78712
Abstract

The Dakota (Design Analysis Kit for Optimization and Terascale Applications) toolkit provides a flexible and extensible interface between simulation codes and iterative analysis methods. Dakota contains algorithms for optimization with gradient and nongradient-based methods; uncertainty quantification with sampling, reliability, and stochastic expansion methods; parameter estimation with nonlinear least squares methods; and sensitivity/variance analysis with design of experiments and parameter study methods. These capabilities may be used on their own or as components within advanced strategies such as surrogate-based optimization, mixed integer nonlinear programming, or optimization under uncertainty. By employing object-oriented design to implement abstractions of the key components required for iterative systems analyses, the Dakota toolkit provides a flexible and extensible problem-solving environment for design and performance analysis of computational models on high performance computers.

This report serves as a reference manual for the commands specification for the Dakota software, providing input overviews, option descriptions, and example specifications.
## Contents

1 Main Page
   1.1 How to Use this Manual .................................................. 7

2 Running Dakota
   2.1 Usage ................................................................. 9
   2.2 Examples ............................................................ 10
   2.3 Execution Phases ...................................................... 10
   2.4 Restarting Dakota Studies ............................................ 11
   2.5 The Dakota Restart Utility ............................................ 12

3 Test Problems
   3.1 Textbook .............................................................. 17
   3.2 Rosenbrock ............................................................ 20

4 Dakota Input Specification
   4.1 Dakota NIDR .......................................................... 21
   4.2 Input Spec Overview .................................................. 21
   4.3 Sample Input Files .................................................... 23
   4.4 Input Spec Summary .................................................. 27

5 Topics Area
   5.1 admin ................................................................. 75
   5.2 dakota.IO .............................................................. 76
   5.3 dakota_concepts ....................................................... 87
   5.4 models ................................................................. 110
   5.5 variables .............................................................. 114
   5.6 responses .............................................................. 120
   5.7 interface ............................................................... 121
   5.8 methods ............................................................... 124
   5.9 advanced_topics ....................................................... 141
   5.10 packages ............................................................. 145

6 Keywords Area
   6.1 environment .......................................................... 158
   6.2 method ................................................................. 190
   6.3 model ................................................................. 1819
   6.4 variables .............................................................. 1964
   6.5 interface .............................................................. 2112
6.6 responses ................................................................. 2165

Bibliographic References .............................. 2256
Chapter 1

Main Page

The Dakota (Design Analysis Kit for Optimization and Terascale Applications) toolkit provides a flexible, extensible interface between analysis codes and iteration methods.

Author


The Reference Manual documents all the input keywords that can appear in a Dakota input file to configure a Dakota study. Its organization closely mirrors the structure of dakota.input.summary. For more information see Dakota Input Specification. For information on software structure, refer to the Developers Manual [3], and for a tour of Dakota features and capabilities, including a tutorial, refer to the User’s Manual (Adams et al., 2010) [4].

1.1 How to Use this Manual

- To learn how to run Dakota from the command line, see Running Dakota
- To learn how to restart Dakota studies, see Restarting Dakota Studies
- To learn about the Dakota restart utility, see The Dakota Restart Utility

To find more information about a specific keyword

1. Use the search box at the top right (currently only finds keyword names)
2. Browse the Keywords tree on the left navigation pane
3. Look at the Dakota Input Specification
4. Navigate through the keyword pages, starting from the Keywords Area
   To find more information about a Dakota related topic
   1. Browse the Topics Area on the left navigation pane
   2. Navigate through the topics pages, starting from the Topics Area

A small number of examples are included (see Sample Input Files) along with a description of the test problems (see Test Problems). A bibliography for the Reference Manual is provided in Bibliographic References

7
Chapter 2

Running Dakota

The Dakota executable file is named dakota (dakota.exe on Windows) and is most commonly run from a terminal or command prompt.

2.1 Usage

If the dakota command is entered at the command prompt without any arguments, a usage message similar to the following appears:

usage: dakota [options and <args>]
   -help (Print this summary)
   -version (Print Dakota version number)
   -input <$val> (REQUIRED Dakota input file $val)
   -output <$val> (Redirect Dakota standard output to file $val)
   -error <$val> (Redirect Dakota standard error to file $val)
   -parser <$val> (Parsing technology: nidr[strict][:dumpfile])
   -no_input_echo (Do not echo Dakota input file)
   -check (Perform input checks)
   -pre_run [$val] (Perform pre-run (variables generation) phase)
   -run [$val] (Perform run (model evaluation) phase)
   -post_run [$val] (Perform post-run (final results) phase)
   -read_restart [$val] (Read an existing Dakota restart file $val)
   -stop_restart <$val> (Stop restart file processing at evaluation $val)
   -write_restart [$val] (Write a new Dakota restart file $val)

Of these command line options, only input is required, and the -input switch can be omitted if the input file name is the final item appearing on the command line (see Examples); all other command-line inputs are optional.

- help prints the usage message above.

- version prints version information for the executable.

- check invokes a dry-run mode in which the input file is processed and checked for errors, but the study is not performed.

- input provides the name of the Dakota input file.

- output and error options provide file names for redirection of the Dakota standard output (stdout) and standard error (stderr), respectively.
• The parser option is for debugging and will not be further described here.

• By default, Dakota will echo the input file to the output stream, but no_input_echo can override this behavior.

• read_restart and write_restart commands provide the names of restart databases to read from and write to, respectively.

• stop_restart command limits the number of function evaluations read from the restart database (the default is all the evaluations) for those cases in which some evaluations were erroneous or corrupted. Restart management is an important technique for retaining data from expensive engineering applications.

• -pre_run, -run, and -post_run instruct Dakota to run one or more execution phases, excluding others. The commands must be followed by filenames as described in Execution Phases.

Command line switches can be abbreviated so long as the abbreviation is unique, so the following are valid, unambiguous specifications: -h, -v, -c, -i, -o, -e, -s, -w, -re, -pr, -ru, and -po and can be used in place of the longer forms of the command line options.

For information on restarting Dakota, see Restarting Dakota Studies and The Dakota Restart Utility.

2.2 Examples

To run Dakota with a particular input file, the following syntax can be used:

```
dakota -i dakota.in
```

or more simply

```
dakota dakota.in
```

This will echo the standard output (stdout) and standard error (stderr) messages to the terminal. To redirect stdout and stderr to separate files, the -o and -e command line options may be used:

```
dakota -i dakota.in -o dakota.out -e dakota.err
```

or

```
dakota -o dakota.out -e dakota.err dakota.in
```

Alternatively, any of a variety of Unix redirection variants can be used. Refer to[7] for more information on Unix redirection. The simplest of these redirects stdout to another file:

```
dakota dakota.in > dakota.out
```

2.3 Execution Phases

Dakota has three execution phases: pre-run, run, and post-run.

• pre-run can be used to generate variable sets

• run (core run) invokes the simulation to evaluate variables, producing responses

• post-run accepts variable/response sets and analyzes the results (for example, calculate correlations from a set of samples). Currently only two modes are supported and only for sampling, parameter study, and DACE methods:

1. pre-run only with optional tabular output of variables:

```
dakota -i dakota.in -pre_run ::myvariables.dat
```

2. post-run only with required tabular input of variables/responses:

```
dakota -i dakota.in -post_run myvarsresponses.dat::
2.4 Restarting Dakota Studies

Dakota is often used to solve problems that require repeatedly running computationally expensive simulation codes. In some cases you may want to repeat an optimization study, but with a tighter final convergence tolerance. This would be costly if the entire optimization analysis had to be repeated. Interruptions imposed by computer usage policies, power outages, and system failures could also result in costly delays. However, Dakota automatically records the variable and response data from all function evaluations so that new executions of Dakota can pick up where previous executions left off. The Dakota restart file (dakota.rst by default) archives the tabulated interface evaluations in a binary format. The primary restart commands at the command line are -read_restart, -write_restart, and -stop_restart.

2.4.1 Writing Restart Files

To write a restart file using a particular name, the -write_restart command line input (may be abbreviated as -w) is used:

```
dakota -i dakota.in -write_restart my_restart_file
```

If no -write_restart specification is used, then Dakota will still write a restart file, but using the default name dakota.rst instead of a user-specified name.

To turn restart recording off, the user may use the restart file keyword, in the interface block. This can increase execution speed and reduce disk storage requirements, but at the expense of a loss in the ability to recover and continue a run that terminates prematurely. This option is not recommended when function evaluations are costly or prone to failure. Please note that using the deactivate restart_file specification will result in a zero length restart file with the default name dakota.rst, which can overwrite an exiting file.

2.4.2 Using Restart Files

To restart Dakota from a restart file, the -read_restart command line input (may be abbreviated as -r) is used:

```
dakota -i dakota.in -read_restart my_restart_file
```

If no -read_restart specification is used, then Dakota will not read restart information from any file (i.e., the default is no restart processing).

To read in only a portion of a restart file, the -stop_restart control (may be abbreviated as -s) is used to specify the number of entries to be read from the database. Note that this integer value corresponds to the restart record processing counter (as can be seen when using the print utility (see The Dakota Restart Utility) which may differ from the evaluation numbers used in the previous run if, for example, any duplicates were detected (since these duplicates are not recorded in the restart file). In the case of a -stop_restart specification, it is usually desirable to specify a new restart file using -write_restart so as to remove the records of erroneous or corrupted function evaluations. For example, to read in the first 50 evaluations from dakota.rst:

```
dakota -i dakota.in -r dakota.rst -s 50 -w dakota_new.rst
```

The dakota_new.rst file will contain the 50 processed evaluations from dakota.rst as well as any new evaluations. All evaluations following the 50th in dakota.rst have been removed from the latest restart record.

2.4.3 Appending to a Restart File

If the -write_restart and -read_restart specifications identify the same file (including the case where -write_restart is not specified and -read_restart identifies dakota.rst), then new evaluations will be appended to the existing restart file.
2.4.4 Working with multiple Restart Files

If the \texttt{-write\_restart} and \texttt{-read\_restart} specifications identify different files, then the evaluations read from the file identified by \texttt{-read\_restart} are first written to the \texttt{-write\_restart} file. Any new evaluations are then appended to the \texttt{-write\_restart} file. In this way, restart operations can be chained together indefinitely with the assurance that all of the relevant evaluations are present in the latest restart file.

2.4.5 How it Works

Dakota’s restart algorithm relies on its duplicate detection capabilities. Processing a restart file populates the list of function evaluations that have been performed. Then, when the study is restarted, it is started from the beginning (not a warm start) and many of the function evaluations requested by the iterator are intercepted by the duplicate detection code. This approach has the primary advantage of restoring the complete state of the iteration (including the ability to correctly detect subsequent duplicates) for all methods/iterators without the need for iterator-specific restart code. However, the possibility exists for numerical round-off error to cause a divergence between the evaluations performed in the previous and restarted studies. This has been rare in practice.

2.5 The Dakota Restart Utility

The Dakota restart utility program provides a variety of facilities for managing restart files from Dakota executions. The executable program name is \texttt{dakota\_restart\_util} and it has the following options, as shown by the usage message returned when executing the utility without any options:

Usage:
\begin{verbatim}
dakota_restart_util command <arg1> [...] --options

dakota_restart_util print <restart_file>

dakota_restart_util to_neutral <restart_file> <neutral_file>

dakota_restart_util from_neutral <neutral_file> <restart_file>

dakota_restart_util to_tabular <restart_file> <text_file> [--custom\_annotated [header] [eval\_id] [interface\_id]


dakota_restart_util remove <double> <old\_restart\_file> <new\_restart\_file>

dakota_restart_util remove\_ids <int\_1> ... <int\_n> <old\_restart\_file> <new\_restart\_file>

dakota_restart_util cat <restart\_file\_1> ... <restart\_file\_n> <new\_restart\_file>
\end{verbatim}

options:
--help show dakota\_restart\_util help message
--custom\_annotated arg tabular file options: header, eval\_id, interface\_id

Several of these functions involve format conversions. In particular, the binary format used for restart files can be converted to ASCII text and printed to the screen, converted to and from a neutral file format, or converted to a tabular format for importing into 3rd-party graphics programs. In addition, a restart file with corrupted data can be repaired by value or id, and multiple restart files can be combined to create a master database.

2.5.1 Print Command

The \texttt{print} option is useful to show contents of a restart file, since the binary format is not convenient for direct inspection. The restart data is printed in full precision, so that exact matching of points is possible for restarted runs or corrupted data removals. For example, the following command

\begin{verbatim}
dakota_restart_util print dakota.rst
\end{verbatim}

results in output similar to the following:

\begin{verbatim}
------------------------------------------
Restart record 1 (evaluation id 1):
------------------------------------------
Parameters: 1.800000000000000e+00 intake\_dia
\end{verbatim}
2.5. THE DAKOTA RESTART UTILITY

1.0000000000000000e+00 flatness

Active response data:
Active set vector = { 3 3 3 3 }
-2.4355973813420619e+00 obj_fn
-4.742848677140930e-01 nln_ineq_con_1
-4.5000000000000000e-01 nln_ineq_con_2
1.397114317029974e-01 nln_ineq_con_3
[ -4.3644298963447897e-01 1.4999999999999999e-01 ] obj_fn gradient
[ 1.3855136437818300e-01 0.0000000000000000e+00 ] nln_ineq_con_1 gradient
[ 0.0000000000000000e+00 1.4999999999999999e-01 ] nln_ineq_con_2 gradient
[ 0.0000000000000000e+00 -1.9485571585149869e-01 ] nln_ineq_con_3 gradient

------------------------------------------
Restart record 2 (evaluation id 2):
------------------------------------------
Parameters:
2.1640000000000000e+00 intake_dia
1.7169994018008317e+00 flatness

Active response data:
Active set vector = { 3 3 3 3 }
-2.4869127192988878e+00 obj_fn
6.9256958799989843e-01 nln_ineq_con_1
-3.4245008972987528e-01 nln_ineq_con_2
8.7142207937157910e-03 nln_ineq_con_3
[ -4.3644298963447897e-01 1.4999999999999999e-01 ] obj_fn gradient
[ 2.9814239699997572e+01 0.0000000000000000e+00 ] nln_ineq_con_1 gradient
[ 0.0000000000000000e+00 1.4999999999999999e-01 ] nln_ineq_con_2 gradient
[ 0.0000000000000000e+00 -1.6998301774282701e-01 ] nln_ineq_con_3 gradient

...<snip>...

Restart file processing completed: 11 evaluations retrieved.

2.5.2 Neutral File Format

A Dakota restart file can be converted to a neutral file format using a command like the following:

dakota_restart_util to_neutral dakota.rst dakota.neu

which results in a report similar to the following:

Writing neutral file dakota.neu
Restart file processing completed: 11 evaluations retrieved.

Similarly, a neutral file can be returned to binary format using a command like the following:

dakota_restart_util from_neutral dakota.neu dakota.rst

which results in a report similar to the following:

Reading neutral file dakota.neu
Writing new restart file dakota.rst
Neutral file processing completed: 11 evaluations retrieved.

The contents of the generated neutral file are similar to the following (from the first two records for the Cylinder example in[4]):

6 7 2 1.8000000000000000e+00 intake_dia 1.0000000000000000e+00 flatness 0 0 0 0
NULL 4 2 1 0 3 3 3 3 1 2 obj_fn nln_ineq_con_1 nln_ineq_con_2 nln_ineq_con_3
-2.4355973813420619e+00 -4.742848677140930e-01 -4.5000000000000000e-01
1.397114317029974e-01 -4.3644298963447897e-01 1.4999999999999999e-01
1.3855136437818300e-01 0.0000000000000000e+00 0.0000000000000000e+00
This format is not intended for direct viewing (print should be used for this purpose). Rather, the neutral file capability has been used in the past for managing portability of restart data across platforms (recent use of more portable binary formats has largely eliminated this need) or for advanced repair of restart records (in cases where the remove command was insufficient).

2.5.3 Tabular Format

Conversion of a binary restart file to a tabular format enables convenient import of this data into 3rd-party post-processing tools such as Matlab, TECplot, Excel, etc. This facility is nearly identical to the output activated by the tabular_data keyword in the Dakota input file specification, but with two important differences:

1. No function evaluations are suppressed as they are with tabular_data (i.e., any internal finite difference evaluations are included).

2. The conversion can be performed later, i.e., for Dakota runs executed previously.

An example command for converting a restart file to tabular format is:

dakota_restart_util to_tabular dakota.rst dakota.m

which results in a report similar to the following:

Writing tabular text file dakota.m
Restart file processing completed: 10 evaluations tabulated.

The contents of the generated tabular file are similar to the following (from the example in the Restart section of[4]). Note that while evaluations resulting from numerical derivative offsets would be reported (as described above), derivatives returned as part of the evaluations are not reported (since they do not readily fit within a compact tabular format):

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0000199640857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>4</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.10011</td>
<td>0.0002004407265</td>
<td>0.259945</td>
<td>0.7602420121</td>
</tr>
<tr>
<td>5</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.109989</td>
<td>0.0001995607255</td>
<td>0.260055</td>
<td>0.7597580121</td>
</tr>
<tr>
<td>6</td>
<td>NO_ID</td>
<td>0.58256179</td>
<td>0.4772224441</td>
<td>0.10505555937</td>
<td>0.1007670171</td>
<td>-0.06353963386</td>
</tr>
<tr>
<td>7</td>
<td>NO_ID</td>
<td>0.5826200462</td>
<td>0.4772224441</td>
<td>0.1050386469</td>
<td>0.1008348962</td>
<td>-0.06356876195</td>
</tr>
<tr>
<td>8</td>
<td>NO_ID</td>
<td>0.5825435359</td>
<td>0.4772224441</td>
<td>0.1050725476</td>
<td>0.1006991449</td>
<td>-0.06351050577</td>
</tr>
<tr>
<td>9</td>
<td>NO_ID</td>
<td>0.58256179</td>
<td>0.4772701663</td>
<td>0.1050283245</td>
<td>0.100743156</td>
<td>-0.06349083333</td>
</tr>
<tr>
<td>10</td>
<td>NO_ID</td>
<td>0.58256179</td>
<td>0.4771747219</td>
<td>0.1050828704</td>
<td>0.1007908783</td>
<td>-0.06358517983</td>
</tr>
</tbody>
</table>

Controlling tabular format: The command-line option --custom_annotated gives control of headers in the resulting tabular file. It supports options

- header: include %-commented header row with labels
- eval_id: include leading column with evaluation ID
- interface_id: include leading column with interface ID
2.5. THE DAKOTA RESTART UTILITY

For example, to recover Dakota 6.0 tabular format, which contained a header row, leading column with evaluation ID, but no interface ID:

```plaintext
dakota_restart_util to_tabular dakota.rst dakota.m --custom.annotated header eval_id
```

Resulting in:

```
%eval_id x1 x2 obj_fn nln.ineq_con_1 nln.ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

### 2.5.4 Concatenation of Multiple Restart Files

In some instances, it is useful to combine restart files into a single master function evaluation database. For example, when constructing a data fit surrogate model, data from previous studies can be pulled in and reused to create a combined data set for the surrogate fit. An example command for concatenating multiple restart files is:

```plaintext
dakota_restart_util cat dakota.rst.1 dakota.rst.2 dakota.rst.3 dakota.rst.all
```

which results in a report similar to the following:

```
Writing new restart file dakota.rst.all
Dakota.rst.1 processing completed: 10 evaluations retrieved.
Dakota.rst.2 processing completed: 110 evaluations retrieved.
Dakota.rst.3 processing completed: 65 evaluations retrieved.
```

The `dakota.rst.all` database now contains 185 evaluations and can be read in for use in a subsequent Dakota study using the `-read restart` option to the `dakota` executable.

### 2.5.5 Removal of Corrupted Data

On occasion, a simulation or computer system failure may cause a corruption of the Dakota restart file. For example, a simulation crash may result in failure of a post-processor to retrieve meaningful data. If 0’s (or other erroneous data) are returned from the user’s analysis driver, then this bad data will get recorded in the restart file. If there is a clear demarcation of where corruption initiated (typical in a process with feedback, such as gradient-based optimization), then use of the `-stop restart` option for the `dakota` executable can be effective in continuing the study from the point immediately prior to the introduction of bad data. If, however, there are interspersed corruptions throughout the restart database (typical in a process without feedback, such as sampling), then the `remove` and `remove_ids` options of `dakota_restart_util` can be useful.

An example of the command syntax for the `remove` option is:

```plaintext
dakota_restart_util remove 2.e-04 dakota.rst dakota.rst.repaired
```

which results in a report similar to the following:

```
Writing new restart file dakota.rst.repaired
Restart repair completed: 65 evaluations retrieved, 2 removed, 63 saved.
```

where any evaluations in `dakota.rst` having an active response function value that matches `2.e-04` within machine precision are discarded when creating `dakota.rst.repaired`.

An example of the command syntax for the `remove_ids` option is:

```plaintext
dakota_restart_util remove_ids 12 15 23 44 57 dakota.rst dakota.rst.repaired
```

which results in a report similar to the following:

```
Writing new restart file dakota.rst.repaired
Restart repair completed: 65 evaluations retrieved, 5 removed, 60 saved.
```
where evaluation ids 12, 15, 23, 44, and 57 have been discarded when creating `dakota.rst.repaired`. An important detail is that, unlike the `-stop_restart` option which operates on restart record numbers, the `remove_ids` option operates on evaluation ids. Thus, removal is not necessarily based on the order of appearance in the restart file. This distinction is important when removing restart records for a run that contained either asynchronous or duplicate evaluations, since the restart insertion order and evaluation ids may not correspond in these cases (asynchronous evaluations have ids assigned in the order of job creation but are inserted in the restart file in the order of job completion, and duplicate evaluations are not recorded which introduces offsets between evaluation id and record number). This can also be important if removing records from a concatenated restart file, since the same evaluation id could appear more than once. In this case, all evaluation records with ids matching the `remove_ids` list will be removed.

If neither of these removal options is sufficient to handle a particular restart repair need, then the fallback position is to resort to direct editing of a neutral file to perform the necessary modifications.
Chapter 3

Test Problems

This page contains additional information about two test problems that are used in Dakota examples throughout the Dakota manuals Textbook and Rosenbrock.

Many of these examples are also used as code verification tests. The examples are run periodically and the results are checked against known solutions. This ensures that the algorithms are correctly implemented.

Additional test problems are described in the User’s Manual.

3.1 Textbook

The two-variable version of the “textbook” test problem provides a nonlinearly constrained optimization test case. It is formulated as:

\[
\begin{align*}
\text{minimize} & \quad f = (x_1 - 1)^4 + (x_2 - 1)^4 \\
\text{subject to} & \quad g_1 = x_1^2 - \frac{x_2}{2} \leq 0 \\
& \quad g_2 = x_2^2 - \frac{x_1}{2} \leq 0 \\
& \quad 0.5 \leq x_1 \leq 5.8 \\
& \quad -2.9 \leq x_2 \leq 2.9
\end{align*}
\]

Contours of this test problem are illustrated in the next two figures.
For the textbook test problem, the unconstrained minimum occurs at \((x_1, x_2) = (1, 1)\). However, the inclusion of the constraints moves the minimum to \((x_1, x_2) = (0.5, 0.5)\). Equation \textbf{textbookform} presents the 2-dimensional
form of the textbook problem. An extended formulation is stated as

$$\begin{align*}
\text{minimize} & \quad f = \sum_{i=1}^{n} (x_i - 1)^4 \\
\text{subject to} & \quad g_1 = x_1^2 - \frac{x_2}{2} \leq 0 \\
& \quad g_2 = x_2^2 - \frac{x_1}{2} \leq 0 \\
& \quad 0.5 \leq x_1 \leq 5.8 \\
& \quad -2.9 \leq x_2 \leq 2.9
\end{align*}$$

where \( n \) is the number of design variables. The objective function is designed to accommodate an arbitrary number of design variables in order to allow flexible testing of a variety of data sets. Contour plots for the \( n = 2 \) case have been shown previously.

For the optimization problem given in Equation tbe, the unconstrained solution (num_nonlinear_inequality_constraints set to zero) for two design variables is:

$$\begin{align*}
x_1 &= 1.0 \\
x_2 &= 1.0
\end{align*}$$

with

$$f^* = 0.0$$

The solution for the optimization problem constrained by \( g_1 \) (num_nonlinear_inequality_constraints set to one) is:

$$\begin{align*}
x_1 &= 0.763 \\
x_2 &= 1.16
\end{align*}$$

with

$$\begin{align*}
f^* &= 0.00388 \\
g_1^* &= 0.0 \text{ (active)}
\end{align*}$$

The solution for the optimization problem constrained by \( g_1 \) and \( g_2 \) (num_nonlinear_inequality_constraints set to two) is:

$$\begin{align*}
x_1 &= 0.500 \\
x_2 &= 0.500
\end{align*}$$

with

$$\begin{align*}
f^* &= 0.125 \\
g_1^* &= 0.0 \text{ (active)} \\
g_2^* &= 0.0 \text{ (active)}
\end{align*}$$

Note that as constraints are added, the design freedom is restricted (the additional constraints are active at the solution) and an increase in the optimal objective function is observed.
3.2 Rosenbrock

The Rosenbrock function\cite{34} is a well-known test problem for optimization algorithms. The standard formulation includes two design variables, and computes a single objective function. This problem can also be posed as a least-squares optimization problem with two residuals to be minimized because the objective function is the sum of squared terms.

**Standard Formulation**

The standard two-dimensional formulation can be stated as

\[
\text{minimize } f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad \text{ (rosenstd)}
\]

Surface and contour plots for this function are shown in the Dakota User’s Manual. The optimal solution is:

\[
\begin{align*}
x_1 &= 1.0 \\
x_2 &= 1.0
\end{align*}
\]

with

\[
f^* = 0.0
\]

**A Least-Squares Optimization Formulation**

This test problem may also be used to exercise least-squares solution methods by recasting the standard problem formulation into:

\[
\text{minimize } f = (f_1)^2 + (f_2)^2 \quad \text{ (rosenls)}
\]

where

\[
f_1 = 10(x_2 - x_1^2) \quad \text{ (rosenr1)}
\]

and

\[
f_2 = 1 - x_1 \quad \text{ (rosenr2)}
\]

are residual terms.

The included analysis driver can handle both formulations. In the Dakota/test directory, the rosenbrock executable (compiled from Dakota_Source/test/rosenbrock.cpp) checks the number of response functions passed in the parameters file and returns either an objective function (as computed from Equation rosenstd) for use with optimization methods or two least squares terms (as computed from Equations rosenr1 - rosenr2) for use with least squares methods. Both cases support analytic gradients of the function set with respect to the design variables. See the User’s Manual for examples of both cases (search for Rosenbrock).
Chapter 4

Dakota Input Specification

4.1 Dakota NIDR

Valid Dakota input is dictated by the NIDR[30] input specification file, dakota.input.nspec. This file is used by a code generator to create parsing system components that are compiled into Dakota. Therefore, dakota.input.nspec and its derived summary, dakota.input.summary, are the definitive source for input syntax, capability options, and optional and required capability sub-parameters for any given Dakota version.

Beginning users may find dakota.input.summary overwhelming or confusing and will likely derive more benefit from adapting example input files to a particular problem. Some examples can be found here: Sample Input Files. Advanced users can master the many input specification possibilities by understanding the structure of the input specification file.

4.2 Input Spec Overview

Refer to the dakota.input.summary file, in Input Spec Summary, for current input specifications.

- The summary describes every keyword including:
  - Whether it is required or optional
  - Whether it takes ARGUMENTS (always required) Additional notes about ARGUMENTS can be found here: Specifying Arguments.
  - Whether it has an ALIAS, or synonym
  - Which additional keywords can be specified to change its behavior

- Additional details and descriptions are described in Keywords Area

- For additional details on NIDR specification logic and rules, refer to[30] (Gay, 2008).

4.2.1 Common Specification Mistakes

Spelling mistakes and omission of required parameters are the most common errors. Some causes of errors are more obscure:

- Documentation of new capability sometimes lags its availability in source and executables, especially stable releases. When parsing errors occur that the documentation cannot explain, reference to the particular input specification used in building the executable, which is installed alongside the executable, will often resolve the errors.
If you want to compare results with those obtained using an earlier version of Dakota (prior to 4.1), your input file for the earlier version must use backslashes to indicate continuation lines for Dakota keywords. For example, rather than

```plaintext
# Comment about the following "responses" keyword...
responses,
  objective_functions = 1
# Comment within keyword "responses"
analytic_gradients
# Another comment within keyword "responses"
no_hessians
```

you would need to write

```plaintext
# Comment about the following "responses" keyword...
responses, \
  objective_functions = 1 \
# Comment within keyword "responses" \
analytic_gradients \
# Another comment within keyword "responses" \
no_hessians
```

with no white space (blanks or tabs) after the `\` character.

In most cases, the NIDR system provides error messages that help the user isolate errors in Dakota input files.

### 4.2.2 Specifying Arguments

Some keywords, such as those providing bounds on variables, have an associated list of values or strings, referred to as arguments.

When the same value should be repeated several times in a row, you can use the notation \(N\ast\text{value}\) instead of repeating the value \(N\) times.

For example

```plaintext
lower_bounds -2.0 -2.0 -2.0
upper_bounds 2.0 2.0 2.0
```

could also be written

```plaintext
lower_bounds 3\ast-2.0
upper_bounds 3\ast 2.0
```

(with optional spaces around the `\ast`).

Another possible abbreviation is for sequences: \(L:S:U\) (with optional spaces around the `:`) is expanded to \(L+L+S\ L+2\ast S\ ...\ U\), and \(L:U\) (with no second colon) is treated as \(L:1:U\).

For example, in one of the test examples distributed with Dakota (test case 2 of `test/dakota_uq_textbook_sop_lhs.in`),

```plaintext
histogram_point = 2
abscissas = 50. 60. 70. 80. 90. 30. 40. 50. 60. 70.
counts = 10 20 30 20 10 10 20 30 20 10
```

could also be written

```plaintext
histogram_point = 2
abscissas = 50 : 10 : 90 30 : 10 : 70
counts = 10:10:30 20 10 10:10:30 20 10
```
4.3 SAMPLE INPUT FILES

Count and sequence abbreviations can be used together. For example

response_levels =
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
can be abbreviated

response_levels =
2*0.0:0.1:1.0

4.3 Sample Input Files

A Dakota input file is a collection of fields from the dakota.input.summary file that describe the problem to be solved by Dakota. Several examples follow.

Sample 1: Optimization

The following sample input file shows single-method optimization of the Textbook Example (see Textbook) using DOT’s modified method of feasible directions. A similar file is available as Dakota/examples/users/textbook_opt_conmin.in.

# Dakota Input File: textbook_opt_conmin.in
environment
  graphics
  tabular_data
    tabular_data_file = 'textbook_opt_conmin.dat'
method
  # dot_mmfd #DOT performs better but may not be available
  conmin_mfd
    max_iterations = 50
    convergence_tolerance = 1e-4
variables
  continuous_design = 2
  initial_point 0.9 1.1
  upper_bounds 5.8 2.9
  lower_bounds 0.5 -2.9
descriptors 'x1' 'x2'
interface
direct
  analysis_driver = 'text_book'
responses
  objective_functions = 1
  nonlinear_inequality_constraints = 2
  numerical_gradients
    method_source dakota
    interval_type central
    fd_gradient_step_size = 1.e-4
  no_hessians

Sample 2: Least Squares (Calibration)

The following sample input file shows a nonlinear least squares (calibration) solution of the Rosenbrock Example (see Rosenbrock) using the NL2SOL method. A similar file is available as Dakota/examples/users/rosen_opt_nls.in

# Dakota Input File: rosen_opt_nls.in
environment
graphics
tabular_data
  tabular_data_file = 'rosen_opt_nls.dat'

method
  max_iterations = 100
  convergence_tolerance = 1e-4
  nl2sol

model
  single

variables
  continuous_design = 2
  initial_point = -1.2 1.0
  lower_bounds = -2.0 -2.0
  upper_bounds = 2.0 2.0
  descriptors = 'x1' 'x2'

interface
  analysis_driver = 'rosenbrock'
    direct

responses
  calibration_terms = 2
  analytic_gradients
    no_hessians

Sample 3: Nondeterministic Analysis
The following sample input file shows Latin Hypercube Monte Carlo sampling using the Textbook Example (see Textbook). A similar file is available as Dakota/test/dakota_uq_textbook_lhs.in.

method,
  sampling,
    samples = 100 seed = 1
    complementary_distribution
      response_levels = 3.6e+11 4.0e+11 4.4e+11
      6.0e+04 6.5e+04 7.0e+04
      3.5e+05 4.0e+05 4.5e+05
    sample_type = lhs

variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
    num_pairs = 3 4
    abscissas = 5 8 10 .1 .2 .3 .4
    counts = 17 21 0 12 24 12 0
    descriptors = 'TF1h' 'TF2h'
  histogram_point_uncertain = 1
    num_pairs = 2
    abscissas = 3 4
4.3. SAMPLE INPUT FILES

```
counts = 1 1
descriptors = 'TF3h'

interface,
   fork async evaluation_concurrency = 5
   analysis_driver = 'text_book'

responses,
   response_functions = 3
   no_gradients
   no_hessians

Sample 4: Parameter Study
The following sample input file shows a 1-D vector parameter study using the Textbook Example (see Textbook). It makes use of the default environment and model specifications, so they can be omitted. A similar file is available in the test directory as Dakota/examples/users/rosen_ps_vector.in.

# Dakota Input File: rosen_ps_vector.in
environment
   graphics
   tabular_data
tabular_data_file = 'rosen_ps_vector.dat'
method
   vector_parameter_study
   final_point = 1.1 1.3
   num_steps = 10
variables
   continuous_design = 2
   initial_point = -0.3 0.2
   descriptors 'x1' "x2"

interface
   analysis_driver = 'rosenbrock'
   direct
responses
   objective_functions = 1
   no_gradients
   no_hessians

Sample 5: Hybrid Strategy
The following sample input file shows a hybrid environment using three methods. It employs a genetic algorithm, pattern search, and full Newton gradient-based optimization in succession to solve the Textbook Example (see Textbook). A similar file is available as Dakota/examples/users/textbook_hybrid_strat.in.

environment
   graphics
   hybrid sequential
   method_list = 'PS' 'PS2' 'NLP'
method
   id_method = 'PS'
   model_pointer = 'M1'
coliny_pattern_search stochastic
   seed = 1234
   initial_delta = 0.1
   threshold_delta = 1.e-4
   solution_accuracy = 1.e-10
```
exploratory_moves basic_pattern
#endif verbose output

method
id_method = 'PS2'
model_pointer = 'M1'
max_function_evaluations = 10
coliny_pattern_search stochastic
seed = 1234
initial_delta = 0.1
threshold_delta = 1.e-4
solution_accuracy = 1.e-10
exploratory_moves basic_pattern
#endif verbose output

method
id_method = 'NLP'
model_pointer = 'M2'
optpp_newton
gradient_tolerance = 1.e-12
convergence_tolerance = 1.e-15
#endif verbose output

model
id_model = 'M1'
single
variables_pointer = 'V1'
interface_pointer = 'I1'
responses_pointer = 'R1'

model
id_model = 'M2'
single
variables_pointer = 'V1'
interface_pointer = 'I1'
responses_pointer = 'R2'

variables
id_variables = 'V1'
continuous_design = 2
initial_point 0.6 0.7
upper_bounds 5.8 2.9
lower_bounds 0.5 -2.9
descriptors 'x1' 'x2'

interface
id_interface = 'I1'
direct
analysis_driver = 'text_book'

responses
id_responses = 'R1'
objective_functions = 1
no_gradients
no_hessians

responses
id_responses = 'R2'
objective_functions = 1
analytic_gradients
analytic_hessians
4.4 Input Spec Summary

This file is derived automatically from dakota.input.nspec, which is used in the generation of parser system files that are compiled into the Dakota executable. Therefore, these files are the definitive source for input syntax, capability options, and associated data inputs. Refer to the Developers Manual information on how to modify the input specification and propagate the changes through the parsing system.

Key features of the input specification and the associated user input files include:

- In the input specification, required individual specifications simply appear, optional individual and group specifications are enclosed in [], required group specifications are enclosed in (), and either-or relationships are denoted by the | symbol. These symbols only appear in dakota.input.nspec; they must not appear in actual user input files.

- Keyword specifications (i.e., environment, method, model, variables, interface, and responses) begin with the keyword possibly preceded by white space (blanks, tabs, and newlines) both in the input specifications and in user input files. For readability, keyword specifications may be spread across several lines. Earlier versions of Dakota (prior to 4.1) required a backslash character (\) at the ends of intermediate lines of a keyword. While such backslashes are still accepted, they are no longer required.

- Some of the keyword components within the input specification indicate that the user must supply INTEGER, REAL, STRING, INTEGERLIST, REALLIST, or STRINGLIST data as part of the specification. In a user input file, the "=" is optional, data in a LIST can be separated by commas or whitespace, and the STRING data are enclosed in single or double quotes (e.g., ‘text_book’ or "text_book").

- In user input files, input is largely order-independent (except for entries in lists of data), case insensitive, and white-space insensitive. Although the order of input shown in the Sample Input Files generally follows the order of options in the input specification, this is not required.

- In user input files, specifications may be abbreviated so long as the abbreviation is unique. For example, the npsol.sqp specification within the method keyword could be abbreviated as npsol, but dot.sqp should not be abbreviated as dot since this would be ambiguous with other DOT method specifications.

- In both the input specification and user input files, comments are preceded by #.

- ALIAS refers to synonymous keywords, which often exist for backwards compatibility. Users are encouraged to use the most current keyword.

```plaintext
KEYWORD01 environment
  [ check ]
  [ output_file STRING ]
  [ error_file STRING ]
  [ read_restart STRING ]
    [ stop_restart INTEGER >= 0 ]
  [ write_restart STRING ]
  [ pre_run ]
    [ input STRING ]
    [ output STRING ]
    [ annotated ]
    [ custom.annotated ]
```
[header]
[eval_id]
[interface_id]

[run]
[input STRING]
[output STRING]

[post_run]
[input STRING]
[annotated]
{custom_annotated}
[header]
[eval_id]
[interface_id]

{freeform}

[output STRING]

[graphics]
[tabular_data ALIAS tabular_graphics_data]
[tabular_data_file ALIAS tabular_graphics_file STRING]
[annotated]
{custom_annotated}
[header]
[eval_id]
[interface_id]

{freeform}

[output_precision INTEGER >= 0]

[results_output]
[results_output_file STRING]

[top_method_pointer ALIAS method_pointer STRING]

KEYWORD12 method

{id_method STRING}

[output]
debug
| verbose
| normal
| quiet
| silent

{final_solutions INTEGER >= 0}

hybrid
(sequential ALIAS uncoupled)
{method_name_list STRINGLIST}

[model_pointer_list STRING]

| method_pointer_list STRINGLIST}
4.4. INPUT SPEC SUMMARY

| Embedded ALIAS coupled
| ( global_method_name STRING
|  [ global_model_pointer STRING ]
| )
| global_method_pointer STRING
( local_method_name STRING
  [ local_model_pointer STRING ]
) | local_method_pointer STRING
( local_search_probability REAL |
| )
| Collaborative
| ( method_name_list STRINGLIST
|  [ model_pointer_list STRING ]
| )
| method_pointer_list STRINGLIST |
| [ iterator_servers INTEGER > 0 ]
| [ iterator_scheduling
|  master |
| peer |
| [ processors_per_iterator INTEGER > 0 ]
| )
| Multi_start
| ( method_name STRING
|  [ model_pointer STRING ]
| )
| method_pointer STRING
[ random_starts INTEGER
  [ seed INTEGER ]
| ]
[ starting_points REALLIST ]
[ iterator_servers INTEGER > 0 ]
[ iterator_scheduling
  master |
  peer |
[ processors_per_iterator INTEGER > 0 ]
| )
| Pareto_set
| ( method_name ALIAS opt_method_name STRING
|  [ model_pointer ALIAS opt_model_pointer STRING ]
| )
| method_pointer ALIAS opt_method_pointer STRING
[ random_weight_sets INTEGER
  [ seed INTEGER ]
| ]
[ weight_sets ALIAS multi_objective_weight_sets REALLIST ]
[ iterator_servers INTEGER > 0 ]
[ iterator_scheduling
  master |
  peer |
[ processors_per_iterator INTEGER > 0 ]
| )
| Branch_and_bound
method_pointer STRING
|
  { method_name STRING
    [ model_pointer STRING ]
  }
  [ scaling ]
|
|
{ surrogate_based_local
  method_pointer ALIAS approx_method_pointer STRING
  | method_name ALIAS approx_method_name STRING
  | model_pointer ALIAS approx_model_pointer STRING
  |
    [ surrogate_based_local
      trust_region
        [ initial_size REAL ]
        [ minimum_size REAL ]
        [ contract_threshold REAL ]
        [ expand_threshold REAL ]
        [ contraction_factor REAL ]
        [ expansion_factor REAL ]
      ]
      [ approx_subproblem
        original_primary
          [ single_objective
          | augmented_lagrangian_objective
          | lagrangian_objective
          ]
        | original_constraints
        | linearized_constraints
        | no_constraints
      ]
      [ merit_function
        penalty_merit
          [ adaptive_penalty_merit
          | lagrangian_merit
          | augmented_lagrangian_merit
          ]
      ]
      [ acceptance_logic
        tr_ratio
          [ filter
          ]
        | constraint_relax
        | homotopy
      ]
      [ max_iterations INTEGER >= 0 ]
      [ convergence_tolerance REAL ]
      [ constraint_tolerance REAL ]
    |
|
{ surrogate_based_global
  method_pointer ALIAS approx_method_pointer STRING
  | method_name ALIAS approx_method_name STRING
  | model_pointer ALIAS approx_model_pointer STRING
  |
    [ max_iterations INTEGER >= 0 ]
    [ convergence_tolerance REAL ]
    [ constraint_tolerance REAL ]
    [ speculative ]
|
{ dot_frcg
    [ max_iterations INTEGER >= 0 ]
    [ convergence_tolerance REAL ]
    [ constraint_tolerance REAL ]
    [ speculative ]
}
4.4. INPUT SPEC SUMMARY

```plaintext
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ linear_inequality_constraint_matrix REALIST ]
[ linear_inequality_lower_bounds REALIST ]
[ linear_inequality_upper_bounds REALIST ]
[ linear_inequality_scale_types STRINGLIST ]
[ linear_inequality_scales REALIST ]
[ linear_equality_constraint_matrix REALIST ]
[ linear_equality_targets REALIST ]
[ linear_equality_scale_types STRINGLIST ]
[ linear_equality_scales REALIST ]
[ model_pointer STRING ]
)
| dot_mmfd
| dot_bfgs
| dot_slp
| dot_sqp
|
| ( dot
| frcg
| mmfd
| bfgs
| slp
| sqp
| [ max_iterations INTEGER >= 0 ]
| [ convergence_tolerance REAL ]
| [ constraint_tolerance REAL ]
| [ speculative ]
| [ max_function_evaluations INTEGER >= 0 ]
| [ scaling ]
| [ linear_inequality_constraint_matrix REALIST ]
| [ linear_inequality_lower_bounds REALIST ]
| [ linear_inequality_upper_bounds REALIST ]
| [ linear_inequality_scale_types STRINGLIST ]
| [ linear_inequality_scales REALIST ]
| [ linear_equality_constraint_matrix REALIST ]
| [ linear_equality_targets REALIST ]
| [ linear_equality_scale_types STRINGLIST ]
| [ linear_equality_scales REALIST ]
| [ model_pointer STRING ]
)
|
| ( conmin_frcg
| [ max_iterations INTEGER >= 0 ]
| [ convergence_tolerance REAL ]
| [ constraint_tolerance REAL ]
| [ speculative ]
| [ max_function_evaluations INTEGER >= 0 ]
| [ scaling ]
| [ linear_inequality_constraint_matrix REALIST ]
| [ linear_inequality_lower_bounds REALIST ]
| [ linear_inequality_upper_bounds REALIST ]
| [ linear_inequality_scale_types STRINGLIST ]
| [ linear_inequality_scales REALIST ]
| [ linear_equality_constraint_matrix REALIST ]
| [ linear_equality_targets REALIST ]
| [ linear_equality_scale_types STRINGLIST ]
| [ linear_equality_scales REALIST ]
| [ model_pointer STRING ]
)
|
| conmin_mfd
| |
```
{ conmin
  frcg
  | mfd
  | [ max_iterations INTEGER >= 0 ]
  | [ convergence_tolerance REAL ]
  | [ constraint_tolerance REAL ]
  | [ speculative ]
  | [ max_function_evaluations INTEGER >= 0 ]
  | [ scaling ]
  | [ linear_inequality_constraint_matrix REALLIST ]
  | [ linear_inequality_lower_bounds REALLIST ]
  | [ linear_inequality_upper_bounds REALLIST ]
  | [ linear_inequality_scale_types STRINGLIST ]
  | [ linear_inequality_scales REALLIST ]
  | [ linear_equality_constraint_matrix REALLIST ]
  | [ linear_equality_targets REALLIST ]
  | [ linear_equality_scale_types STRINGLIST ]
  | [ linear_equality_scales REALLIST ]
  | [ model_pointer STRING ]
)
|
{ dl_solver STRING
  [ linear_inequality_constraint_matrix REALLIST ]
  [ linear_inequality_lower_bounds REALLIST ]
  [ linear_inequality_upper_bounds REALLIST ]
  [ linear_inequality_scale_types STRINGLIST ]
  [ linear_inequality_scales REALLIST ]
  [ linear_equality_constraint_matrix REALLIST ]
  [ linear_equality_targets REALLIST ]
  [ linear_equality_scale_types STRINGLIST ]
  [ linear_equality_scales REALLIST ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
{ npsol_sqp
  | [ verify_level INTEGER ]
  | [ function_precision REAL ]
  | [ linesearch_tolerance REAL ]
  | [ convergence_tolerance REAL ]
  | [ max_iterations INTEGER >= 0 ]
  | [ constraint_tolerance REAL ]
  | [ speculative ]
  | [ max_function_evaluations INTEGER >= 0 ]
  | [ scaling ]
  | [ linear_inequality_constraint_matrix REALLIST ]
  | [ linear_inequality_lower_bounds REALLIST ]
  | [ linear_inequality_upper_bounds REALLIST ]
  | [ linear_inequality_scale_types STRINGLIST ]
  | [ linear_inequality_scales REALLIST ]
  | [ linear_equality_constraint_matrix REALLIST ]
  | [ linear_equality_targets REALLIST ]
  | [ linear_equality_scale_types STRINGLIST ]
  | [ linear_equality_scales REALLIST ]
  | [ model_pointer STRING ]
)
|
{ nlssol_sqp
|
{ stanford
  npsol
  | nlssol
4.4. INPUT SPEC SUMMARY

[ verify_level INTEGER ]
[ function_precision REAL ]
[ linesearch_tolerance REAL ]
[ convergence_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ constraint_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ linear_inequality_constraint_matrix REALLIST ]
[ linear_inequality_lower_bounds REALLIST ]
[ linear_inequality_upper_bounds REALLIST ]
[ linear_inequality_scale_types STRINGLIST ]
[ linear_inequality_scales REALLIST ]
[ linear_equality_constraint_matrix REALLIST ]
[ linear_equality_targets REALLIST ]
[ linear_equality_scale_types STRINGLIST ]
[ linear_equality_scales REALLIST ]
[ model_pointer STRING ]

| ( nlpql_sqp
  | max_iterations INTEGER >= 0 ]
  | convergence_tolerance REAL ]
  | linear_inequality_constraint_matrix REALLIST ]
  | linear_inequality_lower_bounds REALLIST ]
  | linear_inequality_upper_bounds REALLIST ]
  | linear_inequality_scale_types STRINGLIST ]
  | linear_inequality_scales REALLIST ]
  | linear_equality_constraint_matrix REALLIST ]
  | linear_equality_targets REALLIST ]
  | linear_equality_scale_types STRINGLIST ]
  | linear_equality_scales REALLIST ]
  | max_function_evaluations INTEGER >= 0 ]
  | scaling ]
  | model_pointer STRING ]

| ( optpp_cg
  | max_step REAL ]
  | gradient_tolerance REAL ]
  | max_iterations INTEGER >= 0 ]
  | convergence_tolerance REAL ]
  | speculative ]
  | max_function_evaluations INTEGER >= 0 ]
  | scaling ]
  | linear_inequality_constraint_matrix REALLIST ]
  | linear_inequality_lower_bounds REALLIST ]
  | linear_inequality_upper_bounds REALLIST ]
  | linear_inequality_scale_types STRINGLIST ]
  | linear_inequality_scales REALLIST ]
  | linear_equality_constraint_matrix REALLIST ]
  | linear_equality_targets REALLIST ]
  | linear_equality_scale_types STRINGLIST ]
  | linear_equality_scales REALLIST ]
  | model_pointer STRING ]

| ( optpp_q_newton
  | optpp_fd_newton
  | optpp_g_newton
  | optpp_newton
search_method
value_based_line_search
gradient_based_line_search
trust_region
tr_pds

merit_function
el_bakry
argaes_tapia
van_shanno

steplength_to_boundary REAL
centering_parameter REAL
max_step REAL
gradient_tolerance REAL
max_iterations INTEGER >= 0
convergence_tolerance REAL

speculative
max_function_evaluations INTEGER >= 0
scaling
linear_inequality_constraint_matrix REALIST
linear_inequality_lower_bounds REALIST
linear_inequality_upper_bounds REALIST
linear_inequality_scale_types STRINGLIST
linear_inequality_scales REALIST
linear_equality_constraint_matrix REALIST
linear_equality_targets REALIST
linear_equality_scale_types STRINGLIST
linear_equality_scales REALIST
model_pointer STRING

optpp_pds
search_scheme_size INTEGER
max_iterations INTEGER >= 0
convergence_tolerance REAL
max_function_evaluations INTEGER >= 0
scaling
model_pointer STRING

asynch_pattern_search ALIAS coliny_apps
initial_delta REAL
contraction_factor REAL
threshold_delta REAL
solution_target ALIAS solution_accuracy REAL
synchronization
blocking
nonblocking
merit_function
merit_max
merit_max_smooth
merit1
merit1_smooth
merit2
merit2_smooth
merit2_squared

constraint_penalty REAL
smoothing_factor REAL
linear_inequality_constraint_matrix REALIST

4.4. INPUT SPEC SUMMARY

[ linear_inequality_lower_bounds REALIST ]
[ linear_inequality_upper_bounds REALIST ]
[ linear_inequality_scale_types STRINGLIST ]
[ linear_inequality_scales REALIST ]
[ linear_equality_constraint_matrix REALIST ]
[ linear_equality_targets REALIST ]
[ linear_equality_scale_types STRINGLIST ]
[ linear_equality_scales REALIST ]
[ constraint_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]

(mesh_adaptive_search
[ function_precision REAL ]
[ seed INTEGER > 0 ]
[ history_file STRING ]
[ display_format STRING ]
[ variable_neighborhood_search REAL ]
[ neighbor_order INTEGER > 0 ]
[ display_all_evaluations ]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]

(moga
[ fitness_type
  layer_rank
  | domination_count
  ]
[ replacement_type
  elitist
  | roulette_wheel
  | unique_roulette_wheel
  ]
[ below_limit REAL
  [ shrinkage_fraction ALIAS shrinkage_percentage REAL ]
  ]
[ niching_type
  radial REALLIST
  | distance REALLIST
  ]
[ max_designs REALLIST
  [ num_designs INTEGER >= 2 ]
  ]
[ convergence_type
  metric_tracker
  [ percent_change REAL ]
  [ num_generations INTEGER >= 0 ]
  ]
[ postprocessor_type
  orthogonal_distance REALLIST
  ]
[ max_iterations INTEGER >= 0 ]
[ linear_inequality_constraint_matrix REALIST ]
[ linear_inequality_lower_bounds REALIST ]
[ linear_inequality_upper_bounds REALIST ]
[ linear_inequality_scale_types STRINGLIST ]
[ linear_inequality_scales REALLIST ]
[ linear_equality_constraint_matrix REALLIST ]
[ linear_equality_targets REALLIST ]
[ linear_equality_scale_types STRINGLIST ]
[ linear_equality_scales REALLIST ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ population_size INTEGER >= 0 ]
[ log_file STRING ]
[ print_each_pop ]
[ initialization_type
  simple_random
  unique_random
  flat_file STRING
 ]
[ crossover_type
  multi_point_binary INTEGER
  multi_point_parameterized_binary INTEGER
  multi_point_real INTEGER
  { shuffle_random
    [ num_parents INTEGER > 0 ]
    [ num_offspring INTEGER > 0 ]
  }
  [ crossover_rate REAL ]
 ]
[ mutation_type
  bit_random
  replace_uniform
  { offset_normal
    [ offset_cauchy
      [ offset_uniform
        [ mutation_scale REAL ]
      ]
      [ mutation_rate REAL ]
    ]
    [ seed INTEGER > 0 ]
    [ convergence_tolerance REAL ]
    [ model_pointer STRING ]
  ]
| { soga
  [ fitness_type
    [ merit_function ]
  ]
  [ replacement_type
    elitist
    [ favor_feasible
      [ roulette_wheel
        [ unique_roulette_wheel
      ]
    ]
    [ convergence_type
      [ best_fitness_tracker
        [ percent_change REAL ]
      ]
      [ num_generations INTEGER >= 0 ]
    ]
  ]
  [ average_fitness_tracker
    [ percent_change REAL ]
  ]
}
4.4. INPUT SPEC SUMMARY

```plaintext
[ num_generations INTEGER >= 0 ]

[ max_iterations INTEGER >= 0 ]
[ linear_inequality_constraint_matrix REALLIST ]
[ linear_inequality_lower_bounds REALLIST ]
[ linear_inequality_upper_bounds REALLIST ]
[ linear_inequality_scale_types STRINGLIST ]
[ linear_inequality_scales REALLIST ]
[ linear_inequality_constraint_matrix REALLIST ]
[ linear_equality_targets REALLIST ]
[ linear_equality_scale_types STRINGLIST ]
[ linear_equality_scales REALLIST ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ population_size INTEGER >= 0 ]
[ log_file STRING ]
[ print_each_pop ]
[ initialization_type
  simple_random
  | unique_random
  | flat_file STRING ]

[ crossover_type
  multi_point_binary INTEGER
  | multi_point_parameterized_binary INTEGER
  | multi_point_real INTEGER
  |
  shuffle_random
  [ num_parents INTEGER > 0 ]
  [ num_offspring INTEGER > 0 ]
  ]

[ crossover_rate REAL ]

[ mutation_type
  bit_random
  | replace_uniform
  |
  offset_normal
  | offset_cauchy
  | offset_uniform
  [ mutation_scale REAL ]
  ]

[ mutation_rate REAL ]

[ seed INTEGER > 0 ]
[ convergence_tolerance REAL ]
[ model_pointer STRING ]
)

[ coliny_pattern_search
  [ constant_penalty ]
  [ no_expansion ]
  [ expand_after_success INTEGER ]
  [ pattern_basis
coordinate
  | simplex
  ]
  [ stochastic ]
  [ total_pattern_size INTEGER ]
  [ exploratory_moves
    multi_step
```
CHAPTER 4. DAKOTA INPUT SPECIFICATION

| adaptive_pattern | basic_pattern |
| [ synchronization | blocking | nonblocking |
| [ contraction_factor REAL | constraint_penalty REAL | initial_delta REAL | threshold_delta REAL | solution_target ALIAS solution_accuracy REAL | seed INTEGER > 0 | show_misc_options | misc_options STRINGLIST | max_iterations INTEGER >= 0 | convergence_tolerance REAL | max_function_evaluations INTEGER >= 0 | scaling | model_pointer STRING |

| coliny_solis_wets |
| contract_after_failure INTEGER | no_expansion | expand_after_success INTEGER | constant_penalty | contraction_factor REAL | constraint_penalty REAL | initial_delta REAL | threshold_delta REAL | solution_target ALIAS solution_accuracy REAL | seed INTEGER > 0 | show_misc_options | misc_options STRINGLIST | max_iterations INTEGER >= 0 | convergence_tolerance REAL | max_function_evaluations INTEGER >= 0 | scaling | model_pointer STRING |

| coliny_cobyla |
| initial_delta REAL | threshold_delta REAL | solution_target ALIAS solution_accuracy REAL | seed INTEGER > 0 | show_misc_options | misc_options STRINGLIST | max_iterations INTEGER >= 0 | convergence_tolerance REAL | max_function_evaluations INTEGER >= 0 | scaling | model_pointer STRING |

| coliny_direct |
| division | major_dimension | all_dimensions |
| global_balance_parameter REAL |
4.4. INPUT SPEC SUMMARY

[ local_balance_parameter REAL ]
[ max_boxsize_limit REAL ]
[ min_boxsize_limit REAL ]
[ constraint_penalty REAL ]
[ solution_target ALIAS solution_accuracy REAL ]
[ seed INTEGER > 0 ]
[ show_misc_options ]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)

| ( coliny_ea
[ population_size INTEGER > 0 ]
[ initialization_type
  simple_random
  | unique_random
  | flat_file STRING ]
[ fitness_type
  linear_rank
  | merit_function ]
[ replacement_type
  random INTEGER
  | chc INTEGER
  | elitist INTEGER
  [ new_solutions_generated INTEGER ]
  ]
[ crossover_rate REAL ]
[ crossover_type
  two_point
  | blend
  | uniform ]
[ mutation_rate REAL ]
[ mutation_type
  replace_uniform
  |
  { offset_normal
    | offset_cauchy
    | offset_uniform
  [ mutation_scale REAL ]
  [ mutation_range INTEGER ]
  ]
  [ non_adaptive ]
  ]
[ constraint_penalty REAL ]
[ solution_target ALIAS solution_accuracy REAL ]
[ seed INTEGER > 0 ]
[ show_misc_options ]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
( coliny_beta
  beta_solver_name STRING
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

( nl2sol
  [ function_precision REAL ]
  [ absolute_conv_tol REAL ]
  [ x_conv_tol REAL ]
  [ singular_conv_tol REAL ]
  [ singular_radius REAL ]
  [ false_conv_tol REAL ]
  [ initial_trust_radius REAL ]
  [ covariance INTEGER ]
  [ regression_diagnostics ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

( nonlinear_cg
  [ misc_options STRINGLIST ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

( ncsu_direct
  [ solution_target ALIAS solution_accuracy REAL ]
  [ min_boxsize_limit REAL ]
  [ volume_boxsize_limit REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

( genie_opt_darts
  [ genie_direct ]
  [ seed INTEGER > 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

( efficient_global
  [ gaussian_process ALIAS kriging ]
  surfpack
  dakota
4.4. INPUT SPEC SUMMARY

```
use_derivatives

import_build_points_file ALIAS import_points_file STRING
  [ annotated
    [ custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    ]
    [ freeform
      [ active_only ]
    ]
  ]

export_approx_points_file ALIAS export_points_file STRING
  [ annotated
    [ custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    ]
    [ freeform
      [ seed INTEGER > 0 ]
      [ max_iterations INTEGER >= 0 ]
      [ model_pointer STRING ]
    ]
  ]

polynomial_chaos ALIAS nond_polynomial_chaos
  [ p_refinement
    uniform
    [ dimension_adaptive
      sobol
      [ decay
      [ generalized
    ]
    ]
    [ askey
      [ wiener ]
    ]
    [ quadrature_order INTEGERLIST
      [ dimension_preference REALLIST ]
      [ nested
      [ non_nested ]
    ]
  ]

  [ sparse_grid_level INTEGERLIST
    [ restricted
    [ unrestricted ]
    [ dimension_preference REALLIST ]
    [ nested
    [ non_nested ]
  ]

  [ cubature_integrand INTEGER

  [ expansion_order INTEGERLIST
    [ dimension_preference REALLIST ]
    [ basis_type
      tensor_product
    ]
    [ total_order
```
CHAPTER 4. DAKOTA INPUT SPECIFICATION

- [adapted]
  - [advancements INTEGER]
  - [soft_convergence_limit INTEGER]

- [collocation_points INTEGERLIST]
  - [collocation_ratio REAL]
  - [ratio_order REAL]
  - [least_order REAL]
  - [least_squares]
  - [svd]
  - [equality_constrained]

- [orthogonal_matching_pursuit ALIAS omp]
  - [noise_tolerance REALLIST]

- [basis_pursuit ALIAS bp]
  - [basis_pursuit_denoising ALIAS bpdn]
    - [noise_tolerance REALLIST]

- [least_angle_regression ALIAS lars]
  - [noise_tolerance REALLIST]

- [least_absolute_shrinkage ALIAS lasso]
  - [noise_tolerance REALLIST]
  - [l2_penalty REAL]

- [cross_validation]
- [use_derivatives]
- [tensor_grid]
- [reuse_points ALIAS reuse_samples]

- [expansion_samples INTEGERLIST]
  - [reuse_points ALIAS reuse_samples]
  - [incremental_lhs]

- [import_build_points_file ALIAS import_points_file STRING]
  - [annotated]
    - [custom_annotated]
      - [header]
        - [eval_id]
      - [interface_id]

    - [freeform]
      - [active_only]

- [orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli]
  - [collocation_points INTEGERLIST]
    - [cross_validation]
    - [tensor_grid INTEGERLIST]
    - [reuse_points ALIAS reuse_samples]
    - [import_build_points_file ALIAS import_points_file STRING]
      - [annotated]
4.4. INPUT SPEC SUMMARY

| import_expansion_file STRING |
| variance_based_decomp |
| interaction_order INTEGER > 0 |
| drop_tolerance REAL |
| diagonal_covariance |
| full_covariance |
| normalized |
| sample_type |
| lhs |
| random |
| probability_refinement ALIAS sample_refinement |
| import |
| adapt_import |
| mm_adapt_import |
| refinement_samples INTEGER |
| |
| import_approx_points_file STRING |
| annotated |
| custom_annotated |
| header |
| eval_id |
| interface_id |
| |
| freeform |
| active_only |
| |
| export_approx_points_file ALIAS export_points_file STRING |
| annotated |
| custom_annotated |
| header |
| eval_id |
| interface_id |
| |
| freeform |
| |
| export_expansion_file STRING |
| fixed_seed |
| max_iterations INTEGER >= 0 |
| convergence_tolerance REAL |
| reliability_levels REALLIST |
| num_reliability_levels INTEGERLIST |
| |
| response_levels REALLIST |
| num_response_levels INTEGERLIST |
| compute |
probabilities
| reliabilities
| gen_reliabilities
  [ system
  series
| parallel
  ]
  ]
[ distribution
  cumulative
  | complementary
  ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ rng
  mt19937
  | rnum2
  ]
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
)
|
{ stoch_collocation ALIAS nond_stoch_collocation
  [ { p_refinement
      uniform
      |
      ( dimension_adaptive
        sobol
        | generalized
      )
    }
    |
    { h_refinement
      uniform
      |
      ( dimension_adaptive
        sobol
        | generalized
      )
      | local_adaptive
    } |
    piecewise
    | askey
    | wiener }
quadrature_order INTEGERLIST
|
( sparse_grid_level INTEGERLIST
  [ restricted
  | unrestricted ]
  [ nodal
  | hierarchical ]
  )
[ dimension_preference REALLIST ]
[ use_derivatives ]
[ nested
| non_nested ]
4.4. INPUT SPEC SUMMARY

```
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
  full_covariance ]
[ sample_type
  lhs
  random ]
[ probability_refinement ALIAS sample_refinement
  import
  adapt_import
  mm_adapt_import
  [ refinement_samples INTEGER ]
]
[ import_approx_points_file STRING
  [ annotated
    ]
  [ custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  ]
  [ freeform
    [ active_only ]
  ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ annotated
    ]
  [ custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  ]
  [ freeform
    ]
  [ fixed_seed ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ reliability_levels REALLIST
    [ num_reliability_levels INTEGERLIST ]
  ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      reliabilities
      gen_reliabilities
      [ system
        series
        parallel
      ]
    ]
  ]
][ distribution
  cumulative
  complementary
][ probability_levels REALLIST
```
[ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
[ num_gen_reliability_levels INTEGERLIST ]
]
[ rng
  mt19937
  | rnum2
]
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
)
|
( sampling ALIAS nond_sampling
[ sample_type
  random
  | lhs
  |
  { incremental_lhs
    | incremental_random
    | previous_samples INTEGER
  }
]
[ variance_based_decomp
  [ drop_tolerance REAL ]
]
[ backfill ]
[ principal_components
  { percent_variance_explained REAL ]
]
[ fixed_seed ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  compute
  | probabilities
  | reliabilities
  | gen_reliabilities
  [ system
    series
    | parallel
  ]
  |
]
[ distribution
  cumulative
  | complementary
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ rng
  mt19937
  | rnum2
]
4.4. INPUT SPEC SUMMARY

```plaintext
import

import_building_points_file

export_approx_points_file
```

```plaintext
import_building_points_file

import_approx_points_file
```

```plaintext
export_approx_points_file

export_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```

```plaintext
import_approx_points_file

import_approx_points_file
```
[ eval_id ]
[ interface_id ]
]
| ( freeform |
[ response_levels REALLIST
[ num_response_levels INTEGERLIST ]
[ compute probabilities |
[ gen_reliabilities |
[ system series |
[ parallel |
]
]
[ max_iterations INTEGER >= 0 ]
[ distribution cumulative |
[ complementary ]
[ probability_levels REALLIST
[ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
[ num_gen_reliability_levels INTEGERLIST ]
]
[ rng mt19937 |
[ rnum2 ]
]
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
]
|
{ adaptive_sampling ALIAS nond_adaptive_sampling |
[ emulator_samples INTEGER ]
[ fitness_metric predicted_variance |
[ distance |
[ gradient ]
]
[ batch_selection naive |
[ distance_penalty |
[ topology |
[ constant_liar ]
]
[ batch_size INTEGER ]
[ import_build_points_file ALIAS import_points_file STRING |
[ annotated |
[ custom_annotated |
[ header ]
[ eval_id ]
[ interface_id ]
]
]
| ( freeform |
[ active_only ]
}
4.4. Input Spec Summary

- `export_approx_points_file` ALIAS `export_points_file` STRING
- `custom_annotated`
  - `header` |
  - `eval_id` |
  - `interface_id`
- `freeform`
- `response_levels` REALLIST
  - `num_response_levels` INTEGERLIST
- `compute probabilities`
  - `gen_reliabilities`
  - `system`
  - `parallel`
- `misc_options` STRINGLIST
- `max_iterations` INTEGER >= 0
- `distribution`
  - `cumulative` |
  - `complementary`
- `probability_levels` REALLIST
  - `num_probability_levels` INTEGERLIST
- `gen_reliability_levels` REALLIST
  - `num_gen_reliability_levels` INTEGERLIST
- `rng`
  - `mt19937` |
  - `rnum2` |
  - `samples` INTEGER
- `seed` INTEGER > 0
- `model_pointer` STRING

- `pof_darts` ALIAS `nond_pof_darts`
  - `lipschitz`
    - `local` |
    - `global`
  - `emulator_samples` INTEGER
  - `response_levels` REALLIST
    - `num_response_levels` INTEGERLIST
    - `compute probabilities`
      - `gen_reliabilities`
      - `system`
      - `parallel`
    - `distribution`
CHAPTER 4. DAKOTA INPUT SPECIFICATION

cumulative
| complementary
|
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ] ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ] ]
[ rng
  mt19937
  | rnum2
]
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
)
|
( rkd_darts ALIAS nond_rkd_darts
  [ lipachitz
    local
    | global
  ]
  [ emulator_samples INTEGER ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute probabilities
      | gen_reliabilities
        [ system
          series
          | parallel
        ]
    ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ] ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ] ]
  [ rng
    mt19937
    | rnum2
]
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
)
|
( efficient_subspace ALIAS nond_efficient_subspace
  [ emulator_samples INTEGER ]
  [ batch_size INTEGER ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ distribution
    cumulative
  ]
  [ lipschitz
    local
    | global
  ]
  [ emulator_samples INTEGER ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute probabilities
      | gen_reliabilities
        [ system
          series
          | parallel
        ]
    ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ] ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ] ]
  [ rng
    mt19937
    | rnum2
]
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
)
4.4. INPUT SPEC SUMMARY

| complementary |
| probability_levels REALLIST |
| num_probability_levels INTEGERLIST |
| gen_reliability_levels REALLIST |
| num_gen_reliability_levels INTEGERLIST |
| rng mt19937 |
| rnum2 |
| samples INTEGER |
| seed INTEGER > 0 |
| model_pointer STRING |

| global_evidence ALIAS nond_global_evidence |
| sbo |
| ego |
| gaussian_process ALIAS kriging |
| surfpack |
| dakota |
| use_derivatives |
| import_build_points_file ALIAS import_points_file STRING |
| annotated |
| ( custom_annotated |
| header |
| eval_id |
| interface_id |
|
| ( freeform |
| active_only |
|
| export_approx_points_file ALIAS export_points_file STRING |
| annotated |
| ( custom_annotated |
| header |
| eval_id |
| interface_id |
|
| ( freeform |
|
| ea |
| lhs |
| response_levels REALLIST |
| num_response_levels INTEGERLIST |
| compute |
| probabilities |
| gen_reliabilities |
| system |
| series |
| parallel |
|
|
[ distribution
  cumulative
  | complementary
] [ probability_levels REALIST
  [ num_probability_levels INTEGERLIST ] ]
[ gen_reliability_levels REALIST
  [ num_gen_reliability_levels INTEGERLIST ] ]
[ rng
  mt19937
  | rnum2
] [ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
)
)
([ global_interval_est ALIAS nond_global_interval_est
  [ sbo
    | ego
    [ gaussian_process ALIAS kriging
      surfpack
      | dakota
    ]
    | use_derivatives ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ annotated
      | [ custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
      ] [ freeform
        [ active_only ]
      ]
    ]
    | [ export_approx_points_file ALIAS export_points_file STRING
      [ annotated
        | [ custom_annotated
          [ header ]
          [ eval_id ]
          [ interface_id ]
        ] [ freeform
          | ]
      ]
      | [ freeform
        | ]
    ] [ ea
      | lhs ]
  [ rng
    mt19937
    | rnum2
  ] [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ samples INTEGER ]
  [ seed INTEGER > 0 ]}
4.4. INPUT SPEC SUMMARY

[ model_pointer STRING ]
|
| ( bayes_calibration ALIAS nond_bayes_calibration
  ( queso
    [ emulator
      ( gaussian_process ALIAS kriging
        surfpack
        | dakota
        [ emulator_samples INTEGER ]
        [ posterior_adaptive ]
      )
      import_build_points_file ALIAS import_points_file STRING
      [ annotated ]
      | ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
      )
      | ( freeform
        [ active_only ]
      )
    )
  )
  | ( pce
    sparse_grid_level INTEGERLIST
    | expansion_order INTEGERLIST
    | collocation_ratio REAL
    | ( posterior_adaptive
      [ import_build_points_file ALIAS import_points_file STRING
        [ annotated ]
        | ( custom_annotated
          [ header ]
          [ eval_id ]
          [ interface_id ]
        )
        | ( freeform
          [ active_only ]
        )
      )
    )
  )
| ( collocation_points INTEGERLIST
  | ( posterior_adaptive
    [ import_build_points_file ALIAS import_points_file STRING
      [ annotated ]
      | ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
      )
      | ( freeform
        [ active_only ]
      )
    )
  )
)
(sc
sparse_grid_level INTEGERLIST
)
  [ use_derivatives ]
  | 
  [ logit_transform ]
  [ export_chain_points_file STRING
    [ annotated
    | 
    [ custom.annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    ]
    | 
    [ freeform
    | 
    [ dram
      [ delayed_rejection
      | adaptive_metropolis
      | metropolis_hastings
      | multilevel ]
    ]
    [ rng
      mt19937
      | rnum2
    ]
    [ pre_solve
      sqp
      | nip
    ]
    [ proposal_covariance
      { derivatives
        [ proposal_updates INTEGER ]
      }
      | prior
      | 
      [ values REALLIST
        diagonal
        | matrix
      ]
      | 
      ( filename STRING
        diagonal
        | matrix
      )
    ]
  ]
)

( gpmsa
emulator_samples INTEGER
  [ import_build_points_file ALIAS import_points_file STRING
    [ annotated
    | 
    [ custom.annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    ]
    | 
    [ freeform
      [ active_only ]
    | 
    ]
  ]
)
4.4. INPUT SPEC SUMMARY

```plaintext
[ dram
| delayed_rejection
| adaptive_metropolis
| metropolis_hastings
| multilevel ]

[ rng
  mt19937
  | rnum2
]
```

```plaintext
[ pre_solve
  sgp
  | nip
]
```

```plaintext
[ proposal_covariance
  ( derivatives
  [ proposal_updates INTEGER ]
  )
  | prior
  | ( values REALIST
  diagonal
  | matrix
  )
  |
  ( filename STRING
  diagonal
  | matrix
  )
]
```

```plaintext
( wasabi
  [ emulator
    ( gaussian_process ALIAS kriging
    surfpack
    | dakota
    [ emulator_samples INTEGER ]
    [ posterior_adaptive ]
    [ import_build_points_file ALIAS import_points_file STRING
      [ annotated
      ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
        )
      |
      ( freeform
      [ active_only ]
      )
      )
    |
    ( pce
    sparse_grid_level INTEGERLIST
    )
    [ expansion_order INTEGERLIST
    collocation_ratio REAL
    [ posterior_adaptive ]
    [ import_build_points_file ALIAS import_points_file STRING
      [ annotated
      [ custom_annotated
      ```
<table>
<thead>
<tr>
<th>header</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>eval_id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interface_id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freeform</td>
<td></td>
<td></td>
</tr>
<tr>
<td>active_only</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>collocation_points INTEGERLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>posterior_adaptive</td>
</tr>
<tr>
<td>import_build_points_file ALIAS import_points_file STRING</td>
</tr>
<tr>
<td>annotated</td>
</tr>
<tr>
<td>custom_annotated</td>
</tr>
<tr>
<td>header</td>
</tr>
<tr>
<td>eval_id</td>
</tr>
<tr>
<td>interface_id</td>
</tr>
<tr>
<td>freeform</td>
</tr>
<tr>
<td>active_only</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sparse_grid_level INTEGERLIST</th>
</tr>
</thead>
</table>

| use_derivatives |

<table>
<thead>
<tr>
<th>data_distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>gaussian</td>
</tr>
<tr>
<td>means REALLIST</td>
</tr>
<tr>
<td>covariance REALLIST</td>
</tr>
<tr>
<td>diagonal</td>
</tr>
<tr>
<td>matrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>obs_data_filename STRING</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>posterior_density_export_filename STRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>posterior_samples_export_filename STRING</td>
</tr>
<tr>
<td>posterior_samples_import_filename STRING</td>
</tr>
<tr>
<td>generate_posterior_samples</td>
</tr>
<tr>
<td>evaluate_posterior_density</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dream</th>
</tr>
</thead>
<tbody>
<tr>
<td>chains INTEGER &gt;= 3</td>
</tr>
<tr>
<td>num_cr INTEGER &gt;= 1</td>
</tr>
<tr>
<td>crossover_chain_pairs INTEGER &gt;= 0</td>
</tr>
<tr>
<td>gr_threshold REAL &gt; 0.0</td>
</tr>
<tr>
<td>jump_step INTEGER &gt;= 0</td>
</tr>
<tr>
<td>emulator</td>
</tr>
<tr>
<td>gaussian_process ALIAS kriging</td>
</tr>
<tr>
<td>surfpack</td>
</tr>
<tr>
<td>dakota</td>
</tr>
<tr>
<td>emulator_samples INTEGER</td>
</tr>
<tr>
<td>posterior_adaptive</td>
</tr>
</tbody>
</table>
[ import_build_points_file ALIAS import_points_file STRING
  [ annotated
    |
    ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    |
    ( freeform
      [ active_only ]
    )
  ]
  |
  ( pce
    sparse_grid_level INTEGERLIST
    |
    ( expansion_order INTEGERLIST
      collocation_ratio REAL
      [ posterior_adaptive ]
    )
    |
    ( import_build_points_file ALIAS import_points_file STRING
      [ annotated
        |
        ( custom_annotated
          [ header ]
          [ eval_id ]
          [ interface_id ]
        )
        |
        ( freeform
          [ active_only ]
        )
      )
    )
    |
    ( collocation_points INTEGERLIST
      [ posterior_adaptive ]
    )
    |
    ( import_build_points_file ALIAS import_points_file STRING
      [ annotated
        |
        ( custom_annotated
          [ header ]
          [ eval_id ]
          [ interface_id ]
        )
        |
        ( freeform
          [ active_only ]
        )
      )
    )
    |
    ( sc
      sparse_grid_level INTEGERLIST
    )
    |
    ( use_derivatives )
  ]
)]

[ standardized_space ]
[ calibrate_error_multipliers
  one
  per_experiment
  per_response
]
| both
| hyperprior_alphas REALIST
| hyperprior_betas REALIST
|
| convergence_tolerance REAL
| max_iterations INTEGER >= 0
| samples INTEGER
| seed INTEGER > 0
| model_pointer STRING
|
| (dace
| grid
| random
| oas
| lhs
| oa_lhs
| box_behnken
| central_composite
| main_effects
| quality_metrics
| variance_based_decomp
| drop_tolerance REAL
|
| fixed_seed
| symbols INTEGER
| samples INTEGER
| seed INTEGER > 0
| model_pointer STRING
|
| (fsu_cv
| latinize
| variance_metrics
| drop_tolerance REAL
|
| fixed_seed
| trial_type
| grid
| halton
| random
|
| num_trials INTEGER
| max_iterations INTEGER >= 0
| samples INTEGER
| seed INTEGER > 0
| model_pointer STRING
|
| (psuade_moat
| partitions INTEGERLIST
| samples INTEGER
| seed INTEGER > 0
| model_pointer STRING
|
| (local_evidence ALIAS nond_local_evidence
| sqp
| nip
| response_levels REALIST
4.4. INPUT SPEC SUMMARY

[ num_response_levels INTEGERLIST ]
[ compute
  probabilities
  | gen_reliabilities
  [ system
    series
    | parallel
  ]
]

[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
  |
  model_pointer STRING ]
}

( local_interval_est ALIAS nond_local_interval_est
[ sqp
  | nip ]
[ convergence_tolerance REAL ]
[ model_pointer STRING ]
)

( local_reliability ALIAS nond_local_reliability
[ mpp_search
  x_taylor_mean
  | u_taylor_mean
  | x_taylor_mpp
  | u_taylor_mpp
  | x_two_point
  | u_two_point
  | no_approx
  [ sqp
  | nip ]
[ integration
  first_order
  | second_order
  [ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGER ]
  [ seed INTEGER > 0 ]
  ]
)

( response_levels REALLIST
[ num_response_levels INTEGERLIST ]
[ compute
  probabilities
  | reliabilities
  | gen_reliabilities
  [ system
    series
    | parallel
  ]
)]
CHAPTER 4. DAKOTA INPUT SPECIFICATION

[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ distribution
  cumulative
  | complementary
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ model_pointer STRING ]

[ global_reliability ALIAS nond_global_reliability
  x_gaussian_process ALIAS x_kriging
  | u_gaussian_process ALIAS u_kriging
  surfpack
  | dakota ]
[ import_build_points_file ALIAS import_points_file STRING
  [ annotated
    | { custom.annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    ]
    | { freeform
      [ active_only ]
    ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ annotated
    | { custom.annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    ]
    | { freeform
      [ use_derivatives ]
    ]
  ]
[ seed INTEGER > 0 ]
[ rng
  mt19937
  | rnum2 ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
  ]
  [ system
    series
  ]
4.4. INPUT SPEC SUMMARY

| parallel |
| |
| ]
| [ max_iterations INTEGER >= 0 ]
| [ convergence_tolerance REAL ]
| [ distribution
| cumulative |
| complementary |
| [ probability_levels REALLIST |
| [ num_probability_levels INTEGERLIST ] |
| [ gen_reliability_levels REALLIST |
| [ num_gen_reliability_levels INTEGERLIST ] |
| [ model_pointer STRING ] |
|
| ( fsu_quasi_mc
| halton |
| hammersley |
| latinize ] |
| [ quality_metrics ] |
| [ variance_based_decomp |
| [ drop_tolerance REAL ] |
| ]
| [ samples INTEGER ]
| [ fixed_sequence |
| [ sequence_start INTEGERLIST ] |
| [ sequence_leap INTEGERLIST ] |
| [ prime_base INTEGERLIST ] |
| [ max_iterations INTEGER >= 0 ] |
| [ model_pointer STRING ] |
}|
| ( vector_parameter_study
| final_point REALLIST |
| step_vector REALLIST |
| num_steps INTEGER |
| [ model_pointer STRING ] |
}|
| ( list_parameter_study
| list_of_points REALLIST |
| [ import_points_file STRING |
| [ annotated |
| [ custom_annotated |
| [ header ] |
| [ eval_id ] |
| [ interface_id ] |
| ]
| [ freeform |
| [ active_only ] |
| ) |
| [ model_pointer STRING ] |
}|
| ( centered_parameter_study


CHAPTER 4. DAKOTA INPUT SPECIFICATION

step_vector REALIST
steps_per_variable ALIAS deltas_per_variable INTEGERLIST
 [ model_pointer STRING ]

| ( multidim_parameter_study
partitions INTEGERLIST
 [ model_pointer STRING ]
)
|
| ( richardson_extrap
 estimate_order
 | converge_order
 | converge_qoi
 [ refinement_rate REAL ]
 [ convergence_tolerance REAL ]
 [ max_iterations INTEGER >= 0 ]
 [ model_pointer STRING ]
)

KEYWORD model
 [ id_model STRING ]
 [ variables_pointer STRING ]
 [ responses_pointer STRING ]
 [ hierarchical_tagging ]
 [ single
 | interface_pointer STRING ]
 |
| ( surrogate
 [ id_surrogates INTEGERLIST ]
 ( global
 | gaussian_process ALIAS kriging
 | dakota
 [ point_selection ]
 [ trend
 | constant
 | linear
 | reduced_quadratic
 ]
 )
 | ( surfpack
 [ trend
 | constant
 | linear
 | reduced_quadratic
 | quadratic
 ]
 [ optimization_method STRING ]
 [ max_trials INTEGER > 0 ]
 [ nugget REAL > 0 ]
 | find_nugget INTEGER |
 [ correlation_lengths REALLIST ]
 [ export_model
 | filename_prefix STRING ]
 ( formats
 | text_archive ]
 | binary_archive ]
 | algebraic_file ]
 | algebraic_console ]
 )
4.4. INPUT SPEC SUMMARY

```
}]
}
|
| { mars
| [ max_bases INTEGER ]
| [ interpolation
| linear
| cubic
| [ export_model
| [ filename_prefix STRING ]
| formats
| [ text_archive ]
| [ binary_archive ]
}
}
|
| { moving_least_squares
| [ basis_order ALIAS poly_order INTEGER >= 0 ]
| [ weight_function INTEGER ]
| [ export_model
| [ filename_prefix STRING ]
| formats
| [ text_archive ]
| [ binary_archive ]
}
}
|
| { neural_network
| [ max_nodes ALIAS nodes INTEGER ]
| [ range REAL ]
| [ random_weight INTEGER ]
| [ export_model
| [ filename_prefix STRING ]
| formats
| [ text_archive ]
| [ binary_archive ]
| [ algebraic_file ]
| [ algebraic_console ]
}
}
|
| { radial_basis
| [ bases INTEGER ]
| [ max_pts INTEGER ]
| [ min_partition INTEGER ]
| [ max_subsets INTEGER ]
| [ export_model
| [ filename_prefix STRING ]
| formats
| [ text_archive ]
| [ binary_archive ]
| [ algebraic_file ]
| [ algebraic_console ]
}
```
{ polynomial
  basis_order INTEGER >= 0
  | linear
  | quadratic
  | cubic
  [ export_model
    [ filename_prefix STRING ]
  ]
  [ formats
    [ text_archive ]
    [ binary_archive ]
    [ algebraic_file ]
    [ algebraic_console ]
  ]
  ]
  [ domain_decomposition
    [ cell_type STRING ]
    [ support_layers INTEGER ]
    [ discontinuity_detection
      jump_threshold REAL
      gradient_threshold REAL
    ]
  ]
  [ total_points INTEGER
    | minimum_points
    | recommended_points ]
  [ dace_method_pointer STRING
    | actual_model_pointer STRING ]
  [ reuse_points ALIAS reuse_samples
    all
    | region
    | none
  ]
  [ import_build_points_file ALIAS import_points_file ALIAS samples_file STRING
    [ annotated
      | ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
      ]
      | ( freeform
        [ active_only ]
      )
    ]
  ]
  [ export_approx_points_file ALIAS export_points_file STRING
    [ annotated
      | ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
      ]
      | ( freeform
        | [ use_derivatives ]
        [ correction
          zeroth_order
          | first_order
          | second_order
          additive
        ]
      )
    ]
  ]
4.4. INPUT SPEC SUMMARY

| multiplicative
| combined
|
| metrics ALIAS diagnostics STRINGLIST
| cross_validation
| folds INTEGER
| percent REAL |
|
| press |
|
| import_challenge_points_file ALIAS challenge_points_file STRING
| annotated |
| ( custom.annotated |
| header ]
| eval_id ]
| interface_id ]
|
| ( freeform |
| active_only ] |
|
| multipoint |
| tana |
| actual_model_pointer STRING |
|
| local |
| taylor_series |
| actual_model_pointer STRING |
|
| hierarchical |
| low_fidelity_model_pointer STRING |
| high_fidelity_model_pointer STRING |
| correction |
| zeroth_order |
| first_order |
| second_order |
| additive |
| multiplicative |
| combined |
|
| nested |
| optional_interface_pointer STRING |
| optional_interface_responses_pointer STRING |
|
| sub_method_pointer STRING |
| iterator_servers INTEGER > 0 |
| iterator_scheduling |
| master |
| peer |
|
| processors_per_iterator INTEGER > 0 |
| primary_variable_mapping STRINGLIST |
| secondary_variable_mapping STRINGLIST |
| primary_response_mapping REALLIST |
CHAPTER 4. DAKOTA INPUT SPECIFICATION

[ secondary_response_mapping REALLIST ]
)
)

KEYWORD12 variables
[ id_variables STRING ]
[ active
  all
  | design
  | uncertain
  | aleatory
  | epistemic
  | state ]
[ mixed
  | relaxed ]
[ continuous_design INTEGER > 0
  [ initial_point ALIAS cdv_initial_point REALLIST ]
  [ lower_bounds ALIAS cdv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS cdv_upper_bounds REALLIST ]
  [ scale_types ALIAS cdv_scale_types STRINGLIST ]
  [ scales ALIAS cdv_scales REALLIST ]
  [ descriptors ALIAS cdv_descriptors STRINGLIST ]
]
[ discrete_design_range INTEGER > 0
  [ initial_point ALIAS ddv_initial_point INTEGERLIST ]
  [ lower_bounds ALIAS ddv_lower_bounds INTEGERLIST ]
  [ upper_bounds ALIAS ddv_upper_bounds INTEGERLIST ]
  [ descriptors ALIAS ddv_descriptors STRINGLIST ]
]
[ discrete_design_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ categorical STRINGLIST
      [ adjacency_matrix INTEGERLIST ]
    ]
    [ initial_point INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ string INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values STRINGLIST
    [ adjacency_matrix INTEGERLIST ]
    [ initial_point STRINGLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ real INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values REALLIST
    [ categorical STRINGLIST
      [ adjacency_matrix INTEGERLIST ]
    ]
    [ initial_point REALLIST ]
    [ descriptors STRINGLIST ]
  ]
]
[ normal_uncertain INTEGER > 0
  means ALIAS nuv_means REALLIST
  std_deviations ALIAS nuv_std_deviations REALLIST
  [ lower_bounds ALIAS nuv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS nuv_upper_bounds REALLIST ]
]
4.4. INPUT SPEC SUMMARY

[ initial_point REALLIST ]
[ descriptors ALIAS nuv_descriptors STRINGLIST ]

[ lognormal_uncertain INTEGER > 0
  ( lambdas ALIAS lnuv_lambdas REALLIST
    zetas ALIAS lnuv_zetas REALLIST
  )
  ( means ALIAS lnuv_means REALLIST
    std_deviations ALIAS lnuv_std_deviations REALLIST
    error_factors ALIAS lnuv_error_factors REALLIST
  )
  ( lower_bounds ALIAS lnuv_lower_bounds REALLIST ]
  [ initial_point REALLIST ]
  [ descriptors ALIAS lnuv_descriptors STRINGLIST ]
]

[ uniform_uncertain INTEGER > 0
  lower_bounds ALIAS uuv_lower_bounds REALLIST
  upper_bounds ALIAS uuv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS uuv_descriptors STRINGLIST ]
]

[ loguniform_uncertain INTEGER > 0
  lower_bounds ALIAS luuv_lower_bounds REALLIST
  upper_bounds ALIAS luuv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS luuv_descriptors STRINGLIST ]
]

[ triangular_uncertain INTEGER > 0
  modes ALIAS tuv_modes REALLIST
  lower_bounds ALIAS tuv_lower_bounds REALLIST
  upper_bounds ALIAS tuv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS tuv_descriptors STRINGLIST ]
]

[ exponential_uncertain INTEGER > 0
  betas ALIAS euv_betas REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS euv_descriptors STRINGLIST ]
]

[ beta_uncertain INTEGER > 0
  alphas ALIAS buv_alphas REALLIST
  betas ALIAS buv_betas REALLIST
  lower_bounds ALIAS buv_lower_bounds REALLIST
  upper_bounds ALIAS buv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS buv_descriptors STRINGLIST ]
]

[ gamma_uncertain INTEGER > 0
  alphas ALIAS gauv_alphas REALLIST
  betas ALIAS gauv_betas REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS gauv_descriptors STRINGLIST ]
]

[ gumbel_uncertain INTEGER > 0
  alphas ALIAS guuv_alphas REALLIST
  betas ALIAS guuv_betas REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS guuv_descriptors STRINGLIST ]
]

[ frechet_uncertain INTEGER > 0
  [ initial_point REALLIST ]
  [ descriptors ALIAS euv_descriptors STRINGLIST ]
]
alphas ALIAS fuv_alphas REALLIST
betas ALIAS fuv_betas REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS fuv_descriptors STRINGLIST ]
]

[ weibull_uncertain INTEGER > 0
alphas ALIAS wuv_alphas REALLIST
betas ALIAS wuv_betas REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS wuv_descriptors STRINGLIST ]
]

[ histogram_bin_uncertain INTEGER > 0
[ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
abscissas ALIAS huv_bin_abscissas REALLIST
ordinates ALIAS huv_bin_ordinates REALLIST
| counts ALIAS huv_bin_counts REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS huv_bin_descriptors STRINGLIST ]
]

[ poisson_uncertain INTEGER > 0
lambdas REALLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]

[ binomial_uncertain INTEGER > 0
probability_per_trial ALIAS prob_per_trial REALLIST
num_trials INTEGERLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]

[ negative_binomial_uncertain INTEGER > 0
probability_per_trial ALIAS prob_per_trial REALLIST
num_trials INTEGERLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]

[ geometric_uncertain INTEGER > 0
probability_per_trial ALIAS prob_per_trial REALLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]

[ hypergeometric_uncertain INTEGER > 0
total_population INTEGERLIST
selected_population INTEGERLIST
num_drawn INTEGERLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]

[ histogram_point_uncertain
[ integer INTEGER > 0
[ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
abscissas INTEGERLIST
counts REALLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]

[ string INTEGER > 0
[ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
abscissas STRINGLIST
counts REALLIST
[ initial_point STRINGLIST ]
[ descriptors STRINGLIST ]
]
4.4. INPUT SPEC SUMMARY

[ real INTEGER > 0
  [ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
  abscissas REALLIST
  counts REALLIST
  [ initial_point REALLIST ]
  [ descriptors STRINGLIST ]
]

[ uncertain_correlation_matrix REALLIST ]
[ continuous_interval_uncertain ALIAS interval_uncertain INTEGER > 0
  [ num_intervals ALIAS iuv_num_intervals INTEGERLIST ]
  [ interval_probabilities ALIAS iuv_num_intervals integer_probs REALLIST ]
  lower_bounds REALLIST
  upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS iuv_descriptors STRINGLIST ]
]

[ discrete_interval_uncertain ALIAS discrete_uncertain_range INTEGER > 0
  [ num_intervals ALIAS range_num_intervals INTEGERLIST ]
  [ interval_probabilities ALIAS range_interval_probs REALLIST ]
  lower_bounds INTEGERLIST
  upper_bounds INTEGERLIST
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]

[ discrete_uncertain_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ set_probabilities ALIAS set_probs REALLIST ]
    [ categorical STRINGLIST ]
    [ initial_point INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ string INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values STRINGLIST
    [ set_probabilities ALIAS set_probs REALLIST ]
    [ initial_point STRINGLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ real INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values REALLIST
    [ set_probabilities ALIAS set_probs REALLIST ]
    [ categorical STRINGLIST ]
    [ initial_point REALLIST ]
    [ descriptors STRINGLIST ]
  ]
]

[ continuous_state INTEGER > 0
  [ initial_state ALIAS csv_initial_state REALLIST ]
  [ lower_bounds ALIAS csv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS csv_upper_bounds REALLIST ]
  [ descriptors ALIAS csv_descriptors STRINGLIST ]
]

[ discrete_state_range INTEGER > 0
  [ initial_state ALIAS dsv_initial_state INTEGERLIST ]
  [ lower_bounds ALIAS dsv_lower_bounds INTEGERLIST ]
  [ upper_bounds ALIAS dsv_upper_bounds INTEGERLIST ]
  [ descriptors ALIAS dsv_descriptors STRINGLIST ]
]
CHAPTER 4. DAKOTA INPUT SPECIFICATION

[ discrete_state_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ categorical STRINGLIST ]
    [ initial_state INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]

[ string INTEGER > 0
  [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values STRINGLIST
    [ initial_state STRINGLIST ]
    [ descriptors STRINGLIST ]
  ]

[ real INTEGER > 0
  [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values REALLIST
    [ categorical STRINGLIST ]
    [ initial_state REALLIST ]
    [ descriptors STRINGLIST ]
  ]
]

KEYWORD12 interface
  [ id_interface STRING ]
  [ algebraic_mappings STRING ]
  [ analysis_drivers STRINGLIST ]
    [ analysis_components STRINGLIST ]
    [ input_filter STRING ]
    [ output_filter STRING ]
  [ system
    [ fork ]
    [ resulta_file STRING ]
    [ allow_existing_results ]
    [ verbatim ]
    [ aprepro ALIAS dprepro ]
    [ labeled ]
    [ file_tag ]
    [ file_save ]
    [ work_directory
      [ named STRING ]
      [ directory_tag ALIAS dir_tag ]
      [ directory_save ALIAS dir_save ]
      [ link_files STRINGLIST ]
      [ copy_files STRINGLIST ]
      [ replace ]
    ]
  ]

  [ direct
    [ processors_per_analysis INTEGER > 0 ]
  ]

  [ matlab ]

  [ python
    [ numpy ]
  ]

  [ scilab ]

  [ grid ]

  [ failure_capture ]
4.4. INPUT SPEC SUMMARY

abort
| retry INTEGER
| recover REALLIST
| continuation |
| deactivate
| active_set_vector |
| evaluation_cache |
| strict_cache_equality |
| cache_tolerance REAL |
| restart_file |
| |
| asynchronous |
| evaluation_concurrency INTEGER > 0 |
| local_evaluation_scheduling |
| dynamic |
| static |
| analysis_concurrency INTEGER > 0 |
| |
| evaluation_servers INTEGER > 0 |
| |
| evaluation_scheduling |
| master |
| peer |
| dynamic |
| static |
| |
| processors_per_evaluation INTEGER > 0 |
| |
| analysis_servers INTEGER > 0 |
| |
| analysis_scheduling |
| master |
| peer |
| |
KEYWORD12 responses
| id_responses STRING |
| descriptors ALIAS response_descriptors STRINGLIST |
| objective_functions ALIAS num_objective_functions INTEGER >= 0 |
| sense STRINGLIST |
| primary_scale_types ALIAS objective_function_scale_types STRINGLIST |
| primary_scales ALIAS objective_function_scales REALLIST |
| weights ALIAS multi_objective_weights REALLIST |
| nonlinear_inequality_constraints ALIAS num_nonlinear_inequality_constraints INTEGER >= 0 |
| lower_bounds ALIAS nonlinear_inequality_lower_bounds REALLIST |
| upper_bounds ALIAS nonlinear_inequality_upper_bounds REALLIST |
| scale_types ALIAS nonlinear_inequality_scale_types STRINGLIST |
| scales ALIAS nonlinear_inequality_scales REALLIST |
| |
| nonlinear_inequality_constraints ALIAS num_nonlinear_inequality_constraints INTEGER >= 0 |
| targets ALIAS nonlinear_inequality_targets REALLIST |
| scale_types ALIAS nonlinear_inequality_scale_types STRINGLIST |
| scales ALIAS nonlinear_inequality_scales REALLIST |
| |
| scalar_objectives ALIAS num_scalar_objectives INTEGER >= 0 |
| field_objectives ALIAS num_field_objectives INTEGER >= 0 |
| lengths INTEGERLIST |
| num_coordinates_per_field INTEGERLIST |
| read_field_coordinates |
| |
CHAPTER 4. DAKOTA INPUT SPECIFICATION

```plaintext

[ calibration_terms ALIAS least_squares_terms ALIAS num_least_squares_terms INTEGER >= 0 ]
[ scalar_calibration_terms INTEGER >= 0 ]
[ field_calibration_terms INTEGER >= 0 ]
lengths INTEGERLIST
[ num_coordinates_per_field INTEGERLIST ]
[ read_field_coordinates ]

[ primary_scale_types ALIAS calibration_term_scale_types ALIAS least_squares_term_scale_types STRINGLIST ]
[ primary_scales ALIAS calibration_term_scales ALIAS least_squares_term_scales REALLIST ]
[ weights ALIAS calibration_weights ALIAS least_squares_weights REALLIST ]

[ calibration_data
[ num_experiments INTEGER >= 0 ]
[ num_config_variables INTEGER >= 0 ]
[ variance_type STRINGLIST ]
[ scalar_data_file STRING
[ annotated
| { custom_annotated
[ header ]
[ exp_id ]
|}
| { freeform
[ interpolate ]
]

[ calibration_data_file ALIAS least_squares_data_file STRING
[ annotated
| { custom_annotated
[ header ]
[ exp_id ]
|}
| { freeform
[ num_experiments INTEGER >= 0 ]
[ num_config_variables INTEGER >= 0 ]
[ variance_type STRINGLIST ]
]
[ nonlinear_inequality_constraints ALIAS num_nonlinear_inequality_constraints INTEGER >= 0
[ lower_bounds ALIAS nonlinear_inequality_lower_bounds REALLIST ]
[ upper_bounds ALIAS nonlinear_inequality_upper_bounds REALLIST ]
[ scale_types ALIAS nonlinear_inequality_scale_types STRINGLIST ]
[ scales ALIAS nonlinear_inequality_scales REALLIST ]
]
[ nonlinear_equality_constraints ALIAS num_nonlinear_equality_constraints INTEGER >= 0
[ targets ALIAS nonlinear_equality_targets REALLIST ]
[ scale_types ALIAS nonlinear_equality_scale_types STRINGLIST ]
[ scales ALIAS nonlinear_equality_scales REALLIST ]
]
]

[ response_functions ALIAS num_response_functions INTEGER >= 0 ]
[ scalar_responses ALIAS num_scalar_responses INTEGER >= 0 ]
[ field_responses ALIAS num_field_responses INTEGER >= 0 ]
lengths INTEGERLIST
[ num_coordinates_per_field INTEGERLIST ]
[ read_field_coordinates ]
```

4.4. INPUT SPEC SUMMARY

| no_gradients | analytic_gradients |
| mixed_gradients |
| id_numerical_gradients INTEGERLIST |
| id_analytic_gradients INTEGERLIST |
| method_source |
| { dakota |
| ignore_bounds |
| [ relative |
| absolute |
| bounds ] |
| |
| vendor |
| interval_type |
| forward |
| central |
| fd_step_size ALIAS fd_gradient_step_size REALLIST |
|
| numerical_gradients |
| method_source |
| { dakota |
| ignore_bounds |
| [ relative |
| absolute |
| bounds ] |
| |
| vendor |
| interval_type |
| forward |
| central |
| fd_step_size ALIAS fd_gradient_step_size REALLIST |
|
| numerical_hessians |
| fd_step_size ALIAS fd_hessian_step_size REALLIST |
| relative |
| absolute |
| bounds |
| forward |
| central |
|
| quasi_hessians |
| { bfgs |
| damped |
| |
| srl |
|
| analytic_hessians |
|
| mixed_hessians |
| id_numerical_hessians INTEGERLIST |
| fd_step_size ALIAS fd_hessian_step_size REALLIST |
| relative |
| absolute |
| bounds |
[ forward
| central ]
[ id_quasi_hessians INTEGERLIST
  { bfgs
    [ damped ]
  }
  | sr1
]
[ id_analytic_hessians INTEGERLIST ]
Chapter 5

Topics Area

This page introduces the user to the topics used to organize keywords.

- admin
- dakota.IO
- dakota_concepts
- models
- variables
- responses
- interface
- methods
- advanced_topics
- packages

5.1 admin

Description
This is only for management while ref man is under construction

Related Topics
- empty
- problem
- not_yet_reviewed
CHAPTER 5. TOPICS AREA

Related Keywords

5.1.1 empty

Description
This topic tracks the keywords which do not have content in the reference manual

Related Topics
Related Keywords

5.1.2 problem

Description
empty

Related Topics
Related Keywords

5.1.3 not_yet_reviewed

Description
Not yet reviewed.

Related Topics
Related Keywords

5.2 dakota_IO

Description
Keywords and Concepts relating inputs to Dakota and outputs from Dakota

Related Topics
- dakota_inputs
- dakota_output
- file_formats

Related Keywords
- error_file : Base filename for error redirection
- output_file : Base filename for output redirection
- input : Base filename for post-run mode data input
5.2. **DAKOTA_IO**

- **output**: Base filename for post-run mode data output
- **input**: Base filename for pre-run mode data input
- **output**: Base filename for pre-run mode data output
- **read_restart**: Base filename for restart file read
- **stop_restart**: Evaluation ID number at which to stop reading restart file
- **input**: Base filename for run mode data input
- **output**: Base filename for run mode data output
- **write_restart**: Base filename for restart file write

### 5.2.1 dakota_inputs

**Description**

empty

**Related Topics**

- **block**
- **data_import_capabilities**

**Related Keywords**

### 5.2.2 block

**Description**

A block is the highest level of keyword organization in Dakota. There are currently 6 blocks in the Dakota input spec:

**Related Topics**

- **block_identifier**
- **block_pointer**

**Related Keywords**

- **environment**: Top-level settings for Dakota execution
- **interface**: Specifies how function evaluations will be performed in order to map the variables into the responses.
- **method**: Begins Dakota method selection and behavioral settings.
- **model**: Specifies how variables are mapped into a set of responses
- **responses**: Description of the model output data returned to Dakota upon evaluation of an interface.
- **variables**: Specifies the parameter set to be iterated by a particular method.
CHAPTER 5. TOPICS AREA

**block_identifier**

**Description**

empty

**Related Topics**

**Related Keywords**

- `id_interface`: Name the interface block; helpful when there are multiple
- `id_method`: Name the method block; helpful when there are multiple
- `id_model`: Give the model block an identifying name, in case of multiple model blocks
- `id_responses`: Name the response block, helpful when there are multiple

**block_pointer**

**Description**

See `block_pointer` for details about pointers.

**Related Topics**

**Related Keywords**

- `top_method_pointer`: Identify which method leads the Dakota study
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
5.2. DAKOTA IO

- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `global_model_pointer`: Pointer to model used by global method
- `global_method_pointer`: Pointer to global method
- `local_model_pointer`: Pointer to model used by local method
- `local_method_pointer`: Pointer to local method
- `model_pointer_list`: Associate models with method names
- `method_pointer_list`: Pointers to methods to execute sequentially or collaboratively
- `global_model_pointer`: Pointer to model used by global method
- `global_method_pointer`: Pointer to global method
- `local_model_pointer`: Pointer to model used by local method
- `local_method_pointer`: Pointer to local method
- `model_pointer_list`: Associate models with method names
- `method_pointer_list`: Pointers to methods to execute sequentially or collaboratively
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `method_pointer`: Pointer to sub-method to run from each starting point
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `method_pointer`: Pointer to optimization or least-squares sub-method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
5.2. **DAKOTA_IO**

- `method_pointer`: Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- `model_pointer`: Identifier for model block to be used by a method
- `method_pointer`: Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `optional_interface_pointer`: Pointer to interface that provides non-nested responses
- `optional_interface_responses_pointer`: Pointer to responses block that defines non-nested responses
- `sub_method_pointer`: The `sub_method_pointer` specifies the method block for the sub-iterator
- `responses_pointer`: Specify which responses block will be used by this model block
- `interface_pointer`: Interface block pointer for the single model type
- `dace_method_pointer`: Specify a method to gather training data
- `high_fidelity_model_pointer`: Pointer to high fidelity model
- `low_fidelity_model_pointer`: Pointer to low fidelity model
- `actual_model_pointer`: Pointer to specify a "truth" model, from which to construct a surrogate
- `actual_model_pointer`: Pointer to specify a "truth" model, from which to construct a surrogate
- `variables_pointer`: Specify which variables block will be included with this model block
- `id_variables`: Name the variables block; helpful when there are multiple

### 5.2.3 data_import_capabilities

**Description**

empty

**Related Topics**

**Related Keywords**

### 5.2.4 dakota_output

**Description**

empty
Related Topics

Related Keywords

- **graphics**: Display a 2D graphics window of variables and responses
- **output_precision**: Control the output precision
- **results_output**: (Experimental) Write a summary file containing the final results
- **results_output_file**: The base file name of the results file
- **tabular_data**: Write a tabular results file with variable and response history
- **tabular_data_file**: File name for tabular data output
- **output**: Control how much method information is written to the screen and output file

5.2.5 **file_formats**

**Description**

See sections "Inputs to Dakota" and "Outputs from Dakota" in the Dakota User’s Manual[4].

Related Topics

Related Keywords

- **annotated**: Selects annotated tabular file format
- **custom.annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **annotated**: Selects annotated tabular file format
- **custom.annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **annotated**: Selects annotated tabular file format
- **custom.annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **aprepro**: Write parameters files in APREPRO syntax
- **labeled**: Requires correct function value labels in results file
- **aprepro**: Write parameters files in APREPRO syntax
- **labeled**: Requires correct function value labels in results file
- **annotated**: Selects annotated tabular file format
- **custom.annotated**: Selects custom-annotated tabular file format
5.2. **DAKOTA\_IO**

- **freeform** : Selects freeform file format
- **active\_only** : Import only active variables from tabular data file
- **annotated** : Selects annotated tabular file format
- **custom\_annotated** : Selects custom-annotated tabular file format
- **freeform** : Selects freeform file format
- **active\_only** : Import only active variables from tabular data file
- **annotated** : Selects annotated tabular file format
- **custom\_annotated** : Selects custom-annotated tabular file format
- **freeform** : Selects freeform file format
- **active\_only** : Import only active variables from tabular data file
- **annotated** : Selects annotated tabular file format
- **custom\_annotated** : Selects custom-annotated tabular file format
- **freeform** : Selects freeform file format
- **active\_only** : Import only active variables from tabular data file
- **annotated** : Selects annotated tabular file format
- **custom\_annotated** : Selects custom-annotated tabular file format
- **freeform** : Selects freeform file format
- **active\_only** : Import only active variables from tabular data file
- **annotated** : Selects annotated tabular file format
- **custom\_annotated** : Selects custom-annotated tabular file format
- **freeform** : Selects freeform file format
- **active\_only** : Import only active variables from tabular data file
- **annotated** : Selects annotated tabular file format
- **custom\_annotated** : Selects custom-annotated tabular file format
- **freeform** : Selects freeform file format
- **active\_only** : Import only active variables from tabular data file
- **annotated** : Selects annotated tabular file format
- **custom\_annotated** : Selects custom-annotated tabular file format
- **freeform** : Selects freeform file format
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annotated tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annoted tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annoted tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annoted tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
• custom_annotated: Selects custom-annoted tabular file format
• freeform: Selects freeform file format
• active_only: Import only active variables from tabular data file
• annotated: Selects annotated tabular file format
5.2. **DAKOTA_IO**

- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **annotated**: Selects annotated tabular file format
- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **active_only**: Import only active variables from tabular data file
- **annotated**: Selects annotated tabular file format
- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **active_only**: Import only active variables from tabular data file
- **annotated**: Selects annotated tabular file format
- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **active_only**: Import only active variables from tabular data file
- **annotated**: Selects annotated tabular file format
- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **active_only**: Import only active variables from tabular data file
- **annotated**: Selects annotated tabular file format
- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **active_only**: Import only active variables from tabular data file
- **annotated**: Selects annotated tabular file format
- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
- **active_only**: Import only active variables from tabular data file
- **annotated**: Selects annotated tabular file format
- **custom_annotated**: Selects custom-annotated tabular file format
- **freeform**: Selects freeform file format
• **annotated**: Selects annotated tabular file format

• **custom_annotated**: Selects custom-annotated tabular file format

• **freeform**: Selects freeform file format

• **active_only**: Import only active variables from tabular data file

• **annotated**: Selects annotated tabular file format

• **custom_annotated**: Selects custom-annotated tabular file format

• **freeform**: Selects freeform file format

• **active_only**: Import only active variables from tabular data file

• **annotated**: Selects annotated tabular file format

• **custom_annotated**: Selects custom-annotated tabular file format

• **freeform**: Selects freeform file format

• **active_only**: Import only active variables from tabular data file
5.3. DAKOTA_CONCEPTS

Description
Miscellaneous concepts related to Dakota operation

Related Topics
- method_independent_controls
- block
- strategies
- command_line_options
- restarts
- pointers
CHAPTER 5. TOPICS AREA

Related Keywords

5.3.1 method_independent_controls

Description

The <method independent controls> are those controls which are valid for a variety of methods. In some cases, these controls are abstractions which may have slightly different implementations from one method to the next. While each of these controls is not valid for every method, the controls are valid for enough methods that it was reasonable to consolidate the specifications.

Related Topics

- linear_constraints

Related Keywords

- max_iterations : Stopping criterion based on number of iterations
- constraint_tolerance : The maximum allowable value of constraint violation still considered to be feasible
- max_function_evaluations : Stopping criteria based on number of function evaluations
- scaling : Turn on scaling for variables, responses, and constraints
- convergence_tolerance : Stopping criterion based on convergence of the objective function or statistics
- max_iterations : Stopping criterion based on number of iterations
- scaling : Turn on scaling for variables, responses, and constraints
- convergence_tolerance : Stopping criterion based on convergence of the objective function or statistics
- max_function_evaluations : Stopping criteria based on number of function evaluations
- max_iterations : Stopping criterion based on number of iterations
- scaling : Turn on scaling for variables, responses, and constraints
- convergence_tolerance : Stopping criterion based on convergence of the objective function or statistics
- max_function_evaluations : Stopping criteria based on number of function evaluations
- max_iterations : Stopping criterion based on number of iterations
- scaling : Turn on scaling for variables, responses, and constraints
- convergence_tolerance : Stopping criterion based on convergence of the objective function or statistics
- max_function_evaluations : Stopping criteria based on number of function evaluations
- max_iterations : Stopping criterion based on number of iterations
- scaling : Turn on scaling for variables, responses, and constraints
- convergence_tolerance : Stopping criterion based on convergence of the objective function or statistics
- max_function_evaluations : Stopping criteria based on number of function evaluations
5.3. **DAKOTA_CONCEPTS**

- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **constraint_tolerance**: The maximum allowable value of constraint violation still considered to be feasible
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **constraint_tolerance**: The maximum allowable value of constraint violation still considered to be feasible
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **scaling**: Turn on scaling for variables, responses, and constraints
• `constraint_tolerance`: The maximum allowable value of constraint violation still considered to be feasible

• `convergence_tolerance`: Stopping criterion based on convergence of the objective function or statistics

• `max_function_evaluations`: Stopping criteria based on number of function evaluations

• `max_iterations`: Stopping criterion based on number of iterations

• `scaling`: Turn on scaling for variables, responses, and constraints

• `speculative`: Compute speculative gradients

• `constraint_tolerance`: The maximum allowable value of constraint violation still considered to be feasible

• `convergence_tolerance`: Stopping criterion based on convergence of the objective function or statistics

• `max_function_evaluations`: Stopping criteria based on number of function evaluations

• `max_iterations`: Stopping criterion based on number of iterations

• `scaling`: Turn on scaling for variables, responses, and constraints

• `speculative`: Compute speculative gradients

• `constraint_tolerance`: The maximum allowable value of constraint violation still considered to be feasible

• `convergence_tolerance`: Stopping criterion based on convergence of the objective function or statistics

• `max_function_evaluations`: Stopping criteria based on number of function evaluations

• `max_iterations`: Stopping criterion based on number of iterations

• `scaling`: Turn on scaling for variables, responses, and constraints

• `speculative`: Compute speculative gradients
• **constraint_tolerance** : The maximum allowable value of constraint violation still considered to be feasible
• **convergence_tolerance** : Stopping criterion based on convergence of the objective function or statistics
• **max_function_evaluations** : Stopping criteria based on number of function evaluations
• **max_iterations** : Stopping criterion based on number of iterations
• **scaling** : Turn on scaling for variables, responses, and constraints
• **speculative** : Compute speculative gradients
• **max_iterations** : Stopping criterion based on number of iterations
• **convergence_tolerance** : Stopping criterion based on convergence of the objective function or statistics
• **max_function_evaluations** : Stopping criteria based on number of function evaluations
• **max_iterations** : Stopping criterion based on number of iterations
• **final_solutions** : Number of designs returned as the best solutions
• **max_iterations** : Stopping criterion based on number of iterations
• **max_iterations** : Stopping criterion based on number of iterations
• **max_function_evaluations** : Stopping criteria based on number of function evaluations
• **scaling** : Turn on scaling for variables, responses, and constraints
• **max_function_evaluations** : Stopping criteria based on number of function evaluations
• **scaling** : Turn on scaling for variables, responses, and constraints
• **convergence_tolerance** : Stopping criterion based on convergence of the objective function or statistics
• **max_function_evaluations** : Stopping criteria based on number of function evaluations
• **max_iterations** : Stopping criterion based on number of iterations
• **convergence_tolerance** : Stopping criterion based on convergence of the objective function or statistics
• **max_iterations** : Stopping criterion based on number of iterations
• **max_iterations** : Stopping criterion based on number of iterations
• **id_method** : Name the method block; helpful when there are multiple
• **convergence_tolerance** : Stopping criterion based on convergence of the objective function or statistics
• **max_iterations** : Stopping criterion based on number of iterations
• **convergence_tolerance** : Stopping criterion based on convergence of the objective function or statistics
• **convergence_tolerance** : Stopping criterion based on convergence of the objective function or statistics
• **max_iterations** : Stopping criterion based on number of iterations
• **max_function_evaluations** : Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations

- **scaling**: Turn on scaling for variables, responses, and constraints

- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics

- **max_function_evaluations**: Stopping criteria based on number of function evaluations

- **speculative**: Compute speculative gradients

- **constraint_tolerance**: The maximum allowable value of constraint violation still considered to be feasible

- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics

- **max_function_evaluations**: Stopping criteria based on number of function evaluations

- **max_iterations**: Stopping criterion based on number of iterations

- **scaling**: Turn on scaling for variables, responses, and constraints

- **constraint_tolerance**: The maximum allowable value of constraint violation still considered to be feasible

- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
5.3. **DAKOTA_CONCEPTS**

- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
CHAPTER 5. TOPICS AREA

- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **output**: Control how much method information is written to the screen and output file
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_iterations**: Stopping criterion based on number of iterations
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_iterations**: Stopping criterion based on number of iterations
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_iterations**: Stopping criterion based on number of iterations
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **constraint_tolerance**: The maximum allowable value of constraint violation still considered to be feasible
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_function_evaluations**: Stopping criteria based on number of function evaluations
- **max_iterations**: Stopping criterion based on number of iterations
- **scaling**: Turn on scaling for variables, responses, and constraints
- **speculative**: Compute speculative gradients
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_iterations**: Stopping criterion based on number of iterations
- **max_iterations**: Stopping criterion based on number of iterations
- **constraint_tolerance**: The maximum allowable value of constraint violation still considered to be feasible
- **convergence_tolerance**: Stopping criterion based on convergence of the objective function or statistics
- **max_iterations**: Stopping criterion based on number of iterations

5.3.2 **linear_constraints**

**Description**

Many methods use linear equality or inequality constraints.
5.3. DAKOTA CONCEPTS

Related Topics

Related Keywords

- `linear_equality_constraint_matrix`: Define coefficients of the linear equalities
- `linear_equality_scale_types`: Specify how each linear equality constraint is scaled
- `linear_equality_scales`: Define the characteristic values to scale linear equalities
- `linear_equality_targets`: Define target values for the linear equality constraints
- `linear_inequality_constraint_matrix`: Define coefficients of the linear inequality constraints
- `linear_inequality_lower_bounds`: Define lower bounds for the linear inequality constraint
- `linear_inequality_scale_types`: Specify how each linear inequality constraint is scaled
- `linear_inequality_scales`: Define the characteristic values to scale linear inequalities
- `linear_inequality_upper_bounds`: Define upper bounds for the linear inequality constraint
- `linear_equality_constraint_matrix`: Define coefficients of the linear equalities
- `linear_inequality_constraint_matrix`: Define coefficients of the linear inequality constraints
- `linear_inequality_upper_bounds`: Define upper bounds for the linear inequality constraint
- `linear_inequality_scale_types`: Specify how each linear inequality constraint is scaled
- `linear_inequality_scales`: Define the characteristic values to scale linear inequalities
- `linear_inequality_upper_bounds`: Define upper bounds for the linear inequality constraint
- `linear_equality_constraint_matrix`: Define coefficients of the linear equalities
- `linear_equality_target`: Define target values for the linear equality constraints
• `linear.equality.scale_types` : Specify how each linear equality constraint is scaled
• `linear.equality.scales` : Define the characteristic values to scale linear equalities
• `linear.equality.targets` : Define target values for the linear equality constraints
• `linear.inequality.constraint_matrix` : Define coefficients of the linear inequality constraints
• `linear.inequality.lower_bounds` : Define lower bounds for the linear inequality constraint
• `linear.inequality.scale_types` : Specify how each linear inequality constraint is scaled
• `linear.inequality.scales` : Define the characteristic values to scale linear inequalities
• `linear.inequality.upper_bounds` : Define upper bounds for the linear inequality constraint
• `linear.equality.constraint_matrix` : Define coefficients of the linear equalities
• `linear.equality.scale_types` : Specify how each linear equality constraint is scaled
• `linear.equality.scales` : Define the characteristic values to scale linear equalities
• `linear.equality.targets` : Define target values for the linear equality constraints
• `linear.inequality.constraint_matrix` : Define coefficients of the linear inequality constraints
• `linear.inequality.lower_bounds` : Define lower bounds for the linear inequality constraint
• `linear.inequality.scale_types` : Specify how each linear inequality constraint is scaled
• `linear.inequality.scales` : Define the characteristic values to scale linear inequalities
• `linear.inequality.upper_bounds` : Define upper bounds for the linear inequality constraint
5.3. DAKOTA_CONCEPTS

- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_inequality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_inequality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequalities
5.3. DAKOTA CONCEPTS

- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
• **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
• **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
• **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
• **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
• **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
• **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
• **linearEquality_scale_types**: Specify how each linear equality constraint is scaled
• **linearEquality_scales**: Define the characteristic values to scale linear equalities
• **linearEquality_targets**: Define target values for the linear equality constraints
• **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
• **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
• **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
• **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
• **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
• **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
• **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
• **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
• **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
• **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
• **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
• **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
• **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
• **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
• **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
• **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
• **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
• **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
• **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
• **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
5.3. **DAKOTA_CONCEPTS**

- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint
- **linear_equality_constraint_matrix**: Define coefficients of the linear equalities
- **linear_equality_scale_types**: Specify how each linear equality constraint is scaled
- **linear_equality_scales**: Define the characteristic values to scale linear equalities
- **linear_equality_targets**: Define target values for the linear equality constraints
- **linear_inequality_constraint_matrix**: Define coefficients of the linear inequality constraints
- **linear_inequality_lower_bounds**: Define lower bounds for the linear inequality constraint
- **linear_inequality_scale_types**: Specify how each linear inequality constraint is scaled
- **linear_inequality_scales**: Define the characteristic values to scale linear inequalities
- **linear_inequality_upper_bounds**: Define upper bounds for the linear inequality constraint

### 5.3.3 **block**

**Description**

A block is the highest level of keyword organization in Dakota. There are currently 6 blocks in the Dakota input spec:

**Related Topics**

- `block_identifier`
- `block_pointer`
Related Keywords

• environment: Top-level settings for Dakota execution
• interface: Specifies how function evaluations will be performed in order to map the variables into the responses.
• method: Begins Dakota method selection and behavioral settings.
• model: Specifies how variables are mapped into a set of responses
• responses: Description of the model output data returned to Dakota upon evaluation of an interface.
• variables: Specifies the parameter set to be iterated by a particular method.

**block_identifier**

**Description**

empty

**Related Topics**

**Related Keywords**

• id_interface: Name the interface block; helpful when there are multiple
• id_method: Name the method block; helpful when there are multiple
• id_model: Give the model block an identifying name, in case of multiple model blocks
• id_responses: Name the response block, helpful when there are multiple

**block_pointer**

**Description**

See **block_pointer** for details about pointers.

**Related Topics**

**Related Keywords**

• top_method_pointer: Identify which method leads the Dakota study
• model_pointer: Identifier for model block to be used by a method
• model_pointer: Identifier for model block to be used by a method
• model_pointer: Identifier for model block to be used by a method
• model_pointer: Identifier for model block to be used by a method
• method_pointer: Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
• model_pointer: Identifier for model block to be used by a method
5.3. **DAKOTA_CONCEPTS**

- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Pointer to model used by global method
- `method_pointer_list`: Pointers to methods to execute sequentially or collaboratively
- `global_model_pointer`: Pointer to model used by global method
• **global_method_pointer**: Pointer to global method
• **local_model_pointer**: Pointer to model used by local method
• **local_method_pointer**: Pointer to local method
• **model_pointer_list**: Associate models with method names
• **method_pointer_list**: Pointers to methods to execute sequentially or collaboratively
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **method_pointer**: Pointer to sub-method to run from each starting point
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **model_pointer**: Identifier for model block to be used by a method
• **method_pointer**: Pointer to optimization or least-squares sub-method
• **model_pointer**: Identifier for model block to be used by a method
5.3. **DAKOTA_CONCEPTS**

- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `method_pointer`: Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- `model_pointer`: Identifier for model block to be used by a method
- `method_pointer`: Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- `model_pointer`: Identifier for model block to be used by a method
- `model_pointer`: Identifier for model block to be used by a method
- `optional_interface_pointer`: Pointer to interface that provides non-nested responses
- `optional_interface_responses_pointer`: Pointer to responses block that defines non-nested responses
- `sub_method_pointer`: The `sub_method_pointer` specifies the method block for the sub-iterator
- `responses_pointer`: Specify which responses block will be used by this model block
- `interface_pointer`: Interface block pointer for the single model type
- `dace_method_pointer`: Specify a method to gather training data
- `high_fidelity_model_pointer`: Pointer to high fidelity model
- `low_fidelity_model_pointer`: Pointer to low fidelity model
- `actual_model_pointer`: Pointer to specify a "truth" model, from which to construct a surrogate
- `actual_model_pointer`: Pointer to specify a "truth" model, from which to construct a surrogate
- `variables_pointer`: Specify which variables block will be included with this model block
- `id_variables`: Name the variables block; helpful when there are multiple

### 5.3.4 strategies

**Description**

empty

**Related Topics**

- `advanced_strategies`
Related Keywords

5.3.5 advanced_strategies

Description
empty

Related Topics

Related Keywords

5.3.6 command_line_options

Description
empty

Related Topics

Related Keywords

5.3.7 restarts

Description
empty

Related Topics

Related Keywords

5.3.8 pointers

Description
For all pointer specifications, if a pointer string is specified and no corresponding id string is available, Dakota will exit with an error message. If the pointer is optional and no pointer string is specified, then the last specification parsed will be used. It is appropriate to omit optional cross-referencing whenever the relationships are unambiguous due to the presence of only one specification.
5.3. DAKOTA CONCEPTS

Related Topics

- block_pointer
- objective_function_pointer

Related Keywords

5.3.9 block_pointer

Description

See block_pointer for details about pointers.

Related Topics

Related Keywords
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer_list` : Associate models with method names
• `method_pointer_list` : Pointers to methods to execute sequentially or collaboratively
• `global_model_pointer` : Pointer to model used by global method
• `global_method_pointer` : Pointer to global method
• `local_model_pointer` : Pointer to model used by local method
• `local_method_pointer` : Pointer to local method
• `model_pointer_list` : Associate models with method names
• `method_pointer_list` : Pointers to methods to execute sequentially or collaboratively
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
• `model_pointer` : Identifier for model block to be used by a method
5.3. **DAKOTA_CONCEPTS**

- `method_pointer` : Pointer to sub-method to run from each starting point
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `modelPointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `method_pointer` : Pointer to optimization or least-squares sub-method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `method_pointer` : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- `model_pointer` : Identifier for model block to be used by a method
- `method_pointer` : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
• **optional_interface_pointer** : Pointer to interface that provides non-nested responses

• **optional_interface_responses_pointer** : Pointer to responses block that defines non-nested responses

• **sub_method_pointer** : The `sub_method_pointer` specifies the method block for the sub-iterator

• **responses_pointer** : Specify which responses block will be used by this model block

• **interface_pointer** : Interface block pointer for the single model type

• **dace_method_pointer** : Specify a method to gather training data

• **high_fidelity_model_pointer** : Pointer to high fidelity model

• **low_fidelity_model_pointer** : Pointer to low fidelity model

• **actual_model_pointer** : Pointer to specify a "truth" model, from which to construct a surrogate

• **variables_pointer** : Specify which variables block will be included with this model block

• **id_variables** : Name the variables block; helpful when there are multiple

5.3.10 **objective_function_pointer**

**Description**

See `block_pointer` for details about pointers.

**Related Topics**

**Related Keywords**

• **id_analytic_gradients** : Identify which analytical gradient corresponds to which response

• **id_numerical_gradients** : Identify which numerical gradient corresponds to which response

• **id_analytic_hessians** : Identify which analytical Hessian corresponds to which response

• **id_numerical_hessians** : Identify which numerical-Hessian corresponds to which response

• **id_quasi_hessians** : Identify which quasi-Hessian corresponds to which response

5.4 **models**

**Description**

Keywords and Concepts relating to the `model` block
5.4. MODELS

Related Topics

- surrogate_models
- recast_models
- multifidelity_models
- reduced_order_models
- nested_models
- advanced_model_recursion

Related Keywords

5.4.1 surrogate_models

Description

empty

Related Topics

- surrogate_based_optimization_methods

Related Keywords

- point_selection : Enable greedy selection of well-spaced build points
- export_model : Exports surrogate model in user-selected format
- filename_prefix : User-customizable portion of exported model filenames
- formats : Formats for surrogate model export
- algebraic_console : Export surrogate model in algebraic format to the console
- algebraic_file : Export surrogate model in algebraic format to a file
- binary_archive : Export surrogate model to a binary archive file
- text_archive : Export surrogate model to a plain-text archive file
- export_model : Exports surrogate model in user-selected format
- filename_prefix : User-customizable portion of exported model filenames
- formats : Formats for surrogate model export
- binary_archive : Export surrogate model to a binary archive file
- text_archive : Export surrogate model to a plain-text archive file
- metrics : Compute surrogate quality metrics
- cross_validation : Perform k-fold cross validation
- **export_model**: Exports surrogate model in user-selected format
- **filename_prefix**: User-customizable portion of exported model filenames
- **formats**: Formats for surrogate model export
- **binary_archive**: Export surrogate model to a binary archive file
- **text_archive**: Export surrogate model to a plain-text archive file
- **export_model**: Exports surrogate model in user-selected format
- **filename_prefix**: User-customizable portion of exported model filenames
- **formats**: Formats for surrogate model export
- **algebraic_console**: Export surrogate model in algebraic format to the console
- **algebraic_file**: Export surrogate model in algebraic format to a file
- **binary_archive**: Export surrogate model to a binary archive file
- **text_archive**: Export surrogate model to a plain-text archive file
- **max_nodes**: Maximum number of hidden layer nodes
- **random_weight**: (Inactive) Random weight control
- **range**: Range for neural network random weights
- **export_model**: Exports surrogate model in user-selected format
- **filename_prefix**: User-customizable portion of exported model filenames
- **formats**: Formats for surrogate model export
- **algebraic_console**: Export surrogate model in algebraic format to the console
- **algebraic_file**: Export surrogate model in algebraic format to a file
- **binary_archive**: Export surrogate model to a binary archive file
- **text_archive**: Export surrogate model to a plain-text archive file
- **bases**: Initial number of radial basis functions
- **export_model**: Exports surrogate model in user-selected format
- **filename_prefix**: User-customizable portion of exported model filenames
- **formats**: Formats for surrogate model export
- **algebraic_console**: Export surrogate model in algebraic format to the console
- **algebraic_file**: Export surrogate model in algebraic format to a file
- **binary_archive**: Export surrogate model to a binary archive file
- **text_archive**: Export surrogate model to a plain-text archive file
5.4. MODELS

- **max_pts**: Maximum number of RBF CVT points
- **max_subsets**: Number of trial RBF subsets
- **min_partition**: (Inactive) Minimum RBF partition
- **reuse_points**: Surrogate model training data reuse control

### 5.4.2 surrogate-based_optimization_methods

**Description**

empty

**Related Topics**

**Related Keywords**

- **efficient_global**: Global Surrogate Based Optimization, a.k.a. EGO
- **surrogate_based_global**: Global Surrogate Based Optimization
- **surrogate_based_local**: Local Surrogate Based Optimization

### 5.4.3 recast_models

**Description**

empty

**Related Topics**

**Related Keywords**

### 5.4.4 multifidelity_models

**Description**

empty

**Related Topics**

**Related Keywords**

### 5.4.5 reduced_order_models

**Description**

empty

**Related Topics**

**Related Keywords**
5.4.6 nested_models

Description
empty

5.4.7 advanced_model_recursion

Description
empty

Related Topics
• hybrid_and_recursions_logic

5.5 variables

Description
Keywords and concepts relating to the variables block

Related Topics
• variable_domain
• variable_type
5.5. VARIABLES

Related Keywords

5.5.1 variable_domain

Description

Dakota variables can be grouped by their valid domains.

1. Mixed: continuous and discrete variables are treated separately
2. Relaxed: noncategorical discrete variables are relaxed and treated as continuous variables (categorical variables are non-relaxable and remain discrete)

Refer to mixed and relaxed for additional information.

Related Topics

- continuous_variables
- discrete_variables

5.5.2 continuous_variables

Description

This page collects information related to the topic of continuous design, uncertain, and state variables.

Related Topics

Related Keywords

- beta_uncertain: Aleatory uncertain variable - beta
- continuous_design: Continuous design variables; each defined by a real interval
- continuous_interval_uncertain: Epistemic uncertain variable - values from one or more continuous intervals
- continuous_state: Continuous state variables
- exponential_uncertain: Aleatory uncertain variable - exponential
- frechet_uncertain: Aleatory uncertain variable - Frechet
- gamma_uncertain: Aleatory uncertain variable - gamma
- gumbel_uncertain: Aleatory uncertain variable - gumbel
- histogram_bin_uncertain: Aleatory uncertain variable - continuous histogram
- lognormal_uncertain: Aleatory uncertain variable - lognormal
- loguniform_uncertain: Aleatory uncertain variable - loguniform
- normal_uncertain: Aleatory uncertain variable - normal (Gaussian)
• **triangular_uncertain** : Aleatory uncertain variable - triangular
• **uniform_uncertain** : Aleatory uncertain variable - uniform
• **weibull_uncertain** : Aleatory uncertain variable - Weibull

### 5.5.3 discrete_variables

**Description**

This page discusses discrete design, uncertain, and state variables (which have `discrete` in their keyword name) as they have similar specifications. These include:

1. Integer ranges
2. Sets of integers
3. Sets of reals
4. Sets of strings and each is described below.

In addition, some aleatory uncertain variables, e.g., `binomial_uncertain`, are discrete integer-valued random variables specified using parameters. These are described on their individual keyword pages.

**Sets**

Sets of integers, reals, and strings have similar specifications, though different value types. The variables are specified using three keywords:

- **Variable declaration keyword** - specifies the number of variables being defined
- **elements_per_variable** - a list of positive integers specifying how many set members each variable admits
  - Length = # of variables
- **elements** - a list of the permissible integer values in ALL sets, concatenated together.
  - Length = sum of `elements_per_variable`, or an integer multiple of number of variables
  - The order is very important here.
  - The list is partitioned according to the values of `elements_per_variable`, and each partition is assigned to a variable.

- The ordering of `elements_per_variable`, and the partitions of `elements` must match the strings from descriptors

For string variables, each string element value must be quoted and may contain alphanumeric, dash, underscore, and colon. White space, quote characters, and backslash/metacharacters are not permitted. Examples are given on the pages:

- discrete design set `integer`
- discrete design set `real`
- discrete design set `string`
- discrete uncertain set `integer`
5.5. VARIABLES

- discrete uncertain set real
- discrete uncertain set string

Range
For discrete variables defined by range(s), the `lower_bounds` and `upper_bounds` restrict the permissible values. For design variables, this constrains the feasible design space and is frequently used to prevent nonphysical designs. This is a discrete interval variable that may take any integer value within bounds (e.g., \([1, 4]\), allowing values of 1, 2, 3, or 4). For some variable types, each variable is can be defined by multiple ranges.

Examples are given on the pages:
- `discrete_interval_uncertain`

Related Topics
- `binomial_uncertain`: Aleatory uncertain discrete variable - binomial
- `discrete_design_range`: Discrete design variables; each defined by an integer interval
- `discrete_design_set`: Set-valued discrete design variables
- `integer`: Integer-valued discrete design variables
- `real`: Real-valued discrete design variables
- `string`: String-valued discrete design set variables
- `discrete_interval_uncertain`: Epistemic uncertain variable - values from one or more discrete intervals
- `discrete_state_range`: Discrete state variables; each defined by an integer interval
- `discrete_state_set`: Set-valued discrete state variables
- `integer`: Discrete state variables, each defined by a set of permissible integers
- `real`: Discrete state variables, each defined by a set of permissible real numbers
- `string`: String-valued discrete state set variables
- `discrete_uncertain_set`: Set-valued discrete uncertain variables
- `integer`: Discrete, epistemic uncertain variable - integers within a set
- `real`: Discrete, epistemic uncertain variable - real numbers within a set
- `string`: Discrete, epistemic uncertain variable - strings within a set
- `geometric_uncertain`: Aleatory uncertain discrete variable - geometric
- `histogram_point_uncertain`: Aleatory uncertain variable - discrete histogram
- `hypergeometric_uncertain`: Aleatory uncertain discrete variable - hypergeometric
- `negative_binomial_uncertain`: Aleatory uncertain discrete variable - negative binomial
- `poisson_uncertain`: Aleatory uncertain discrete variable - Poisson
5.5.4 variable_type

Description
Dakota variables can be grouped by their type, including all, design, uncertain, aleatory, epistemic, or state. There are certain situations where the user may want to explicitly control the subset of variables that is considered active for a certain Dakota method, and override the default alignments between methods and variable types. Refer to active for additional information.

Related Topics
- design_variables
- aleatory_uncertain_variables
- epistemic_uncertain_variables
- state_variables

Related Keywords
5.5.5 design_variables

Description
Design variables are those variables which are modified for the purposes of computing an optimal design.

The most common type of design variables encountered in engineering applications are of the continuous type. These variables may assume any real value within their bounds. All but a handful of the optimization algorithms in Dakota support continuous design variables exclusively.

Related Topics

Related Keywords

5.5.6 aleatory_uncertain_variables

Description
Aleatory uncertainty is also known as inherent variability, irreducible uncertainty, or randomness.

Aleatory uncertainty is predominantly characterized using probability theory. This is the only option implemented in Dakota.
5.5. VARIABLES

Related Topics

Related Keywords

- beta_uncertain : Aleatory uncertain variable - beta
- binomial_uncertain : Aleatory uncertain discrete variable - binomial
- exponential_uncertain : Aleatory uncertain variable - exponential
- frechet_uncertain : Aleatory uncertain variable - Frechet
- gamma_uncertain : Aleatory uncertain variable - gamma
- geometric_uncertain : Aleatory uncertain discrete variable - geometric
- gumbel_uncertain : Aleatory uncertain variable - gumbel
- histogram_bin_uncertain : Aleatory uncertain variable - continuous histogram
- histogram_point_uncertain : Aleatory uncertain variable - discrete histogram
- hypergeometric_uncertain : Aleatory uncertain discrete variable - hypergeometric
- lognormal_uncertain : Aleatory uncertain variable - lognormal
- loguniform_uncertain : Aleatory uncertain variable - loguniform
- negative_binomial_uncertain : Aleatory uncertain discrete variable - negative binomial
- normal_uncertain : Aleatory uncertain variable - normal (Gaussian)
- poisson_uncertain : Aleatory uncertain discrete variable - Poisson
- triangular_uncertain : Aleatory uncertain variable - triangular
- uniform_uncertain : Aleatory uncertain variable - uniform
- weibull_uncertain : Aleatory uncertain variable - Weibull

5.5.7 epistemic_uncertain_variables

Description

Epistemic uncertainty is uncertainty due to lack of knowledge.

In Dakota, epistemic uncertainty is characterized by interval analysis or the Dempster-Shafer theory of evidence.

Note that epistemic uncertainty can also be modeled with probability density functions - similarly to aleatory uncertainty Dakota does not support this capability.
5.5.8 state variables

Description
State variables provide a convenient mechanism for managing additional model parameterizations such as mesh density, simulation convergence tolerances, and time step controls.

Only parameter studies and design of experiments methods will iterate on state variables. The initial state is used as the only value for the state variable for all other methods, unless active state is invoked.

If a method iterates on a state variable, the variable is treated as a design variable with the given bounds, or as a uniform uncertain variable with the given bounds.

If the state variable is defined only by its bounds, and the method does NOT iterate on state variables, then the initial state must be inferred.

Related Topics

Related Keywords
- continuous_state : Continuous state variables
- discrete_state_range : Discrete state variables; each defined by an integer interval
- discrete_state_set : Set-valued discrete state variables
- integer : Discrete state variables, each defined by a set of permissible integers
- real : Discrete state variables, each defined by a set of permissible real numbers
- string : String-valued discrete state set variables

5.6 responses

Description
Keywords and concepts relating to the responses block
5.7. INTERFACE

Related Topics

- response_types

Related Keywords

5.6.1 response_types

Description

The specification must be one of three types:

1. objective and constraint functions
2. calibration (least squares) terms and constraint functions
3. a generic response functions specification.

These correspond to (a) optimization, (b) deterministic (least squares) or stochastic (Bayesian) inversion, and (c) general-purpose analyzer methods such as parameter studies, DACE, and UQ methods, respectively. Refer to responses for additional details and examples.

Related Topics

5.7 interface

Description

Keywords and Concepts relating to the interface block, which is used to connect Dakota to external analysis codes (simulations, etc.)

Related Topics

- simulation_file_management
- workflow_management
- advanced_simulation_interfaces

Related Keywords

5.7.1 simulation_file_management

Description

empty
Related Topics
Related Keywords

5.7.2 workflow_management

Description
empty

Related Topics
Related Keywords

5.7.3 advanced_simulation_interfaces

Description
empty

Related Topics
Related Keywords
• simulation_failure
• concurrency_and_parallelism

Related Keywords
simulation_failure

Description
empty

Related Topics
Related Keywords
concurrency_and_parallelism

Description
empty

Related Topics
Related Keywords

• processors_per_analysis: Specify the number of processors per analysis when Dakota is run in parallel
• analysis_scheduling: Specify the scheduling of concurrent analyses when Dakota is run in parallel
• master: Specify a dedicated master partition for parallel analysis scheduling
• peer: Specify a peer partition for parallel analysis scheduling
5.7. INTERFACE

- **analysis_servers**: Specify the number of analysis servers when Dakota is run in parallel.
- **asynchronous**: Specify analysis driver concurrency, when Dakota is run in serial.
- **analysis_concurrency**: Limit the number of analysis drivers within an evaluation that Dakota will schedule.
- **evaluation_concurrency**: Determine how many concurrent evaluations Dakota will schedule.
- **local_evaluation_scheduling**: Control how local asynchronous jobs are scheduled.
- **master**: Specify a dedicated master partition for parallel evaluation scheduling.
- **peer**: Specify a peer partition for parallel evaluation scheduling.
- **dynamic**: Specify dynamic scheduling in a peer partition when Dakota is run in parallel.
- **static**: Specify static scheduling in a peer partition when Dakota is run in parallel.
- **evaluation_servers**: Specify the number of evaluation servers when Dakota is run in parallel.
- **processors_per_evaluation**: Specify the number of processors per evaluation server when Dakota is run in parallel.
- **iterator_scheduling**: Specify the scheduling of concurrent iterators when Dakota is run in parallel.
- **master**: Specify a dedicated master partition for parallel iterator scheduling.
- **peer**: Specify a peer partition for parallel evaluation scheduling.
- **iterator_servers**: Specify the number of iterator servers when Dakota is run in parallel.
- **processors_per_iterator**: Specify the number of processors per iterator server when Dakota is run in parallel.
- **iterator_scheduling**: Specify the scheduling of concurrent iterators when Dakota is run in parallel.
- **master**: Specify a dedicated master partition for parallel iterator scheduling.
- **peer**: Specify a peer partition for parallel evaluation scheduling.
- **iterator_servers**: Specify the number of iterator servers when Dakota is run in parallel.
- **processors_per_iterator**: Specify the number of processors per iterator server when Dakota is run in parallel.
- **iterator_scheduling**: Specify the scheduling of concurrent iterators when Dakota is run in parallel.
- **master**: Specify a dedicated master partition for parallel iterator scheduling.
- **peer**: Specify a peer partition for parallel evaluation scheduling.
- **iterator_servers**: Specify the number of iterator servers when Dakota is run in parallel.
- **processors_per_iterator**: Specify the number of processors per iterator server when Dakota is run in parallel.
5.8 methods

Description

Keywords and Concepts relating to the method block, including discussion of the different methods and algorithms available in Dakota

Related Topics

- parameter_studies
- sensitivity_analysis_and_design_of_experiments
- uncertainty_quantification
- optimization_and_calibration

Related Keywords

5.8.1 parameter_studies

Description

Parameter studies employ deterministic designs to explore the effect of parametric changes within simulation models, yielding one form of sensitivity analysis. They can help assess simulation characteristics such as smoothness, multi-modality, robustness, and nonlinearity, which affect the choice of algorithms and controls in follow-on optimization and UQ studies.

Dakota’s parameter study methods compute response data sets at a selection of points in the parameter space. These points may be specified as a vector, a list, a set of centered vectors, or a multi-dimensional grid. Capability overviews and examples of the different types of parameter studies are provided in the Users Manual [4].

With the exception of output verbosity (a setting of silent will suppress some parameter study diagnostic output), Dakota’s parameter study methods do not make use of the method independent controls. Therefore, the parameter study documentation which follows is limited to the method dependent controls for the vector, list, centered, and multidimensional parameter study methods.

Related Topics

Related Keywords

5.8.2 sensitivity_analysis_and_design_of_experiments

Description

empty
5.8. METHODS

Related Topics
- design_and_analysis_of_computer_experiments
- sampling

Related Keywords

5.8.3 design_and_analysis_of_computer_experiments

Description
Design and Analysis of Computer Experiments (DACE) methods compute response data sets at a selection of points in the parameter space. Three libraries are provided for performing these studies: DDACE, FSU/Dace, and PSUADE. The design of experiments methods do not currently make use of any of the method independent controls.

Related Topics
- dace : Design and Analysis of Computer Experiments
- fsu_cvt : Design of Computer Experiments - Centroidal Voronoi Tessellation
- fsu_quasi_mcs : Design of Computer Experiments - Quasi-Monte Carlo sampling
- hammersley : Use Hammersley sequences
- psuade_moa : Morris One-at-a-Time

5.8.4 sampling

Description
Sampling techniques are selected using the sampling method selection. This method generates sets of samples according to the probability distributions of the uncertain variables and maps them into corresponding sets of response functions, where the number of samples is specified by the samples integer specification. Means, standard deviations, coefficients of variation (COVs), and 95% confidence intervals are computed for the response functions. Probabilities and reliabilities may be computed for response_levels specifications, and response levels may be computed for either probability_levels or reliability_levels specifications (refer to the Method Commands chapter in the Dakota Reference Manual[5] for additional information).

Currently, traditional Monte Carlo (MC) and Latin hypercube sampling (LHS) are supported by Dakota and are chosen by specifying sample_type as random or lhs. In Monte Carlo sampling, the samples are selected randomly according to the user-specified probability distributions. Latin hypercube sampling is a stratified sampling technique for which the range of each uncertain variable is divided into \( N_s \) segments of equal probability, where \( N_s \) is the number of samples requested. The relative lengths of the segments are determined by the nature of the specified probability distribution (e.g., uniform has segments of equal width, normal has small segments near the mean and larger segments in the tails). For each of the uncertain variables, a sample is selected randomly from each of these equal probability segments. These \( N_s \) values for each of the individual parameters are then combined in a shuffling operation to create a set of \( N_s \) parameter vectors with a specified correlation structure. A feature of the resulting sample set is that every row and column in the hypercube of partitions has exactly one sample. Since the total number of samples is exactly equal to the number of partitions used for each uncertain
variable, an arbitrary number of desired samples is easily accommodated (as compared to less flexible approaches in which the total number of samples is a product or exponential function of the number of intervals for each variable, i.e., many classical design of experiments methods).

Advantages of sampling-based methods include their relatively simple implementation and their independence from the scientific disciplines involved in the analysis. The main drawback of these techniques is the large number of function evaluations needed to generate converged statistics, which can render such an analysis computationally very expensive, if not intractable, for real-world engineering applications. LHS techniques, in general, require fewer samples than traditional Monte Carlo for the same accuracy in statistics, but they still can be prohibitively expensive. For further information on the method and its relationship to other sampling techniques, one is referred to the works by McKay, et al.[59], Iman and Shortencarier[53], and Helton and Davis[46]. Note that under certain separability conditions associated with the function to be sampled, Latin hypercube sampling provides a more accurate estimate of the mean value than does random sampling. That is, given an equal number of samples, the LHS estimate of the mean will have less variance than the mean value obtained through random sampling.

Related Topics

Related Keywords

- importance_sampling: Importance sampling
- sampling: Randomly samples variables according to their distributions

5.8.5 uncertainty_quantification

Description

Dakota provides a variety of methods for propagating both aleatory and epistemic uncertainty.

At a high level, uncertainty quantification (UQ) or nondeterministic analysis is the process of characterizing input uncertainties, forward propagating these uncertainties through a computational model, and performing statistical or interval assessments on the resulting responses. This process determines the effect of uncertainties and assumptions on model outputs or results. In Dakota, uncertainty quantification methods specifically focus on the forward propagation part of the process, where probabilistic or interval information on parametric inputs are mapped through the computational model to assess statistics or intervals on outputs. For an overview of these approaches for engineering applications, consult[42].

UQ is related to sensitivity analysis in that the common goal is to gain an understanding of how variations in the parameters affect the response functions of the engineering design problem. However, for UQ, some or all of the components of the parameter vector, are considered to be uncertain as specified by particular probability distributions (e.g., normal, exponential, extreme value), or other uncertainty structures. By assigning specific distributional structure to the inputs, distributional structure for the outputs (i.e., response statistics) can be inferred. This migrates from an analysis that is more \emph{qualitative} in nature, in the case of sensitivity analysis, to an analysis that is more rigorously \emph{quantitative}.

UQ methods are often distinguished by their ability to propagate aleatory or epistemic input uncertainty characterizations, where aleatory uncertainties are irreducible variabilities inherent in nature and epistemic uncertainties are reducible uncertainties resulting from a lack of knowledge. Since sufficient data is generally available for aleatory uncertainties, probabilistic methods are commonly used for computing response distribution statistics based on input probability distribution specifications. Conversely, for epistemic uncertainties, any use of probability distributions is based on subjective knowledge rather than objective data, and we may alternatively explore nonprobabilistic methods based on interval specifications.

Dakota contains capabilities for performing nondeterministic analysis with both types of input uncertainty. These UQ methods have been developed by Sandia Labs, in conjunction with collaborators in academia[31],[32],[21],[79].
The aleatory UQ methods in Dakota include various sampling-based approaches (e.g., Monte Carlo and Latin Hypercube sampling), local and global reliability methods, and stochastic expansion (polynomial chaos expansions and stochastic collocation) approaches. The epistemic UQ methods include local and global interval analysis and Dempster-Shafer evidence theory. These are summarized below and then described in more depth in subsequent sections of this chapter. Dakota additionally supports mixed aleatory/epistemic UQ via interval-valued probability, second-order probability, and Dempster-Shafer theory of evidence. These involve advanced model recursions and are described in Section.

Dakota contains capabilities for performing nondeterministic analysis with both types of input uncertainty. These UQ methods have been developed by Sandia Labs, in conjunction with collaborators in academia[31],[32],[21],[79].

The choice of uncertainty quantification method depends on how the input uncertainty is characterized, the computational budget, and the desired output accuracy. The recommendations for UQ methods are summarized in Table and are discussed in the remainder of the section.

TODO: Put table in Doxygen if still needed

Related Topics

- aleatory_uncertainty_quantification_methods
- epistemic_uncertainty_quantification_methods
- variable_support

Related Keywords

- adaptive_sampling : (Experimental) Build a GP surrogate and refine it adaptively
- efficient_subspace : (Experimental) efficient subspace method (ESM)
- global_interval_est : Interval analysis using global optimization methods
- global_reliability : Global reliability methods
- gpais : Gaussian Process Adaptive Importance Sampling
- importance_sampling : Importance sampling
- local_interval_est : Interval analysis using local optimization
- local_reliability : Local reliability method
- mpp_search : Specify which MPP search option to use
- pof_darts : Probability-of-Failure (POF) darts is a novel method for estimating the probability of failure based on random sphere-packing.
- rkd_darts : Recursive k-d (RKD) Darts: Recursive Hyperplane Sampling for Numerical Integration of High-Dimensional Functions.
- sampling : Randomly samples variables according to their distributions
5.8.6 aleatory_uncertainty_quantification_methods

Description
Aleatory uncertainty is also known as inherent variability, irreducible uncertainty, or randomness. Aleatory uncertainty is typically characterized using probability theory.

Related Topics
- sampling
- reliability_methods
- stochastic_expansion_methods

Related Keywords
- importance_sampling : Importance sampling

Description
Sampling techniques are selected using the sampling method selection. This method generates sets of samples according to the probability distributions of the uncertain variables and maps them into corresponding sets of response functions, where the number of samples is specified by the samples integer specification. Means, standard deviations, coefficients of variation (COVs), and 95% confidence intervals are computed for the response functions. Probabilities and reliabilities may be computed for response_levels specifications, and response levels may be computed for either probability_levels or reliability_levels specifications (refer to the Method Commands chapter in the Dakota Reference Manual[5] for additional information).

Currently, traditional Monte Carlo (MC) and Latin hypercube sampling (LHS) are supported by Dakota and are chosen by specifying sample_type as random or lhs. In Monte Carlo sampling, the samples are selected randomly according to the user-specified probability distributions. Latin hypercube sampling is a stratified sampling technique for which the range of each uncertain variable is divided into \( N_s \) segments of equal probability, where \( N_s \) is the number of samples requested. The relative lengths of the segments are determined by the nature of the specified probability distribution (e.g., uniform has segments of equal width, normal has small segments near the mean and larger segments in the tails). For each of the uncertain variables, a sample is selected randomly from each of these equal probability segments. These \( N_s \) values for each of the individual parameters are then combined in a shuffling operation to create a set of \( N_s \) parameter vectors with a specified correlation structure. A feature of the resulting sample set is that every row and column in the hypercube of partitions has exactly one sample. Since the total number of samples is exactly equal to the number of partitions used for each uncertain variable, an arbitrary number of desired samples is easily accommodated (as compared to less flexible approaches in which the total number of samples is a product or exponential function of the number of intervals for each variable, i.e., many classical design of experiments methods).

Advantages of sampling-based methods include their relatively simple implementation and their independence from the scientific disciplines involved in the analysis. The main drawback of these techniques is the large number of function evaluations needed to generate converged statistics, which can render such an analysis computationally very expensive, if not intractable, for real-world engineering applications. LHS techniques, in general, require fewer samples than traditional Monte Carlo for the same accuracy in statistics, but they still can be prohibitively expensive. For further information on the method and its relationship to other sampling techniques, one is referred to the works by McKay, et al.[59], Iman and Shortencarier[53], and Helton and Davis[46]. Note that under certain
5.8. METHODS

separability conditions associated with the function to be sampled, Latin hypercube sampling provides a more accurate estimate of the mean value than does random sampling. That is, given an equal number of samples, the LHS estimate of the mean will have less variance than the mean value obtained through random sampling.

Related Topics

Related Keywords

• importance_sampling : Importance sampling
• sampling : Randomly samples variables according to their distributions

reliability_methods

Description

Reliability methods provide an alternative approach to uncertainty quantification which can be less computationally demanding than sampling techniques. Reliability methods for uncertainty quantification are based on probabilistic approaches that compute approximate response function distribution statistics based on specified uncertain variable distributions. These response statistics include response mean, response standard deviation, and cumulative or complementary cumulative distribution functions (CDF/CCDF). These methods are often more efficient at computing statistics in the tails of the response distributions (events with low probability) than sampling based approaches since the number of samples required to resolve a low probability can be prohibitive.

The methods all answer the fundamental question: “Given a set of uncertain input variables, \(X\), and a scalar response function, \(g\), what is the probability that the response function is below or above a certain level, \(\tilde{z}\)?” The former can be written as \(P[g(X) \leq \tilde{z}] = F_g(\tilde{z})\) where \(F_g(\tilde{z})\) is the cumulative distribution function (CDF) of the uncertain response \(g(X)\) over a set of response levels. The latter can be written as \(P[g(X) > \tilde{z}]\) and defines the complementary cumulative distribution function (CCDF).

This probability calculation involves a multi-dimensional integral over an irregularly shaped domain of interest, \(D\), where \(g(X) < z\) as displayed in Figure figUQ05 for the case of two variables. The reliability methods all involve the transformation of the user-specified uncertain variables, \(X\), with probability density function, \(p(x_1, x_2)\), which can be non-normal and correlated, to a space of independent Gaussian random variables, \(u\), possessing a mean value of zero and unit variance (i.e., standard normal variables). The region of interest, \(D\), is also mapped to the transformed space to yield, \(D_u\), where \(g(U) < z\) as shown in Figure figUQ06. The Nataf transformation[17], which is identical to the Rosenblatt transformation[72] in the case of independent random variables, is used in Dakota to accomplish this mapping. This transformation is performed to make the probability calculation more tractable. In the transformed space, probability contours are circular in nature as shown in Figure figUQ06 unlike in the original uncertain variable space, Figure figUQ05. Also, the multi-dimensional integrals can be approximated by simple functions of a single parameter, \(\beta\), called the reliability index. \(\beta\) is the minimum Euclidean distance from the origin in the transformed space to the response surface. This point is also known as the most probable point (MPP) of failure. Note, however, the methodology is equally applicable for generic functions, not simply those corresponding to failure criteria; this nomenclature is due to the origin of these methods within the disciplines of structural safety and reliability. Note that there are local and global reliability methods. The majority of the methods available are local, meaning that a local optimization formulation is used to locate one MPP. In contrast, global methods can find multiple MPPs if they exist.

Related Topics

Related Keywords

• global_reliability : Global reliability methods
• **u_gaussian_process**: Create GP surrogate in u-space
• **x_gaussian_process**: Create GP surrogate in x-space
• **local_reliability**: Local reliability method
• **mpp_search**: Specify which MPP search option to use
• **integration**: Integration approach
• **first_order**: First-order integration scheme
• **probability_refinement**: Allow refinement of probability and generalized reliability results using importance sampling
• **second_order**: Second-order integration scheme
• **no_approx**: Perform MPP search on original response functions (use no approximation)
• **u_taylor_mean**: Form Taylor series approximation in "u-space" at variable means
• **u_taylor_mpp**: U-space Taylor series approximation with iterative updates
• **u_two_point**: Predict MPP using Two-point Adaptive Nonlinear Approximation in "u-space"
• **x_taylor_mean**: Form Taylor series approximation in "x-space" at variable means
• **x_taylor_mpp**: X-space Taylor series approximation with iterative updates
• **x_two_point**: Predict MPP using Two-point Adaptive Nonlinear Approximation in "x-space"
• **probability_refinement**: Allow refinement of probability and generalized reliability results using importance sampling
• **probability_refinement**: Allow refinement of probability and generalized reliability results using importance sampling

**stochastic_expansion_methods**

**Description**

The development of these techniques mirrors that of deterministic finite element analysis utilizing the notions of projection, orthogonality, and weak convergence[31],[32]. Rather than estimating point probabilities, they form an approximation to the functional relationship between response functions and their random inputs, which provides a more complete uncertainty representation for use in multi-code simulations. Expansion methods include polynomial chaos expansions (PCE), which employ multivariate orthogonal polynomials that are tailored to representing particular input probability distributions, and stochastic collocation (SC), which employs multivariate interpolation polynomials. For PCE, expansion coefficients may be evaluated using a spectral projection approach (based on sampling, tensor-product quadrature, Smolyak sparse grid, or cubature methods for numerical integration) or a regression approach (least squares or compressive sensing). For SC, interpolants are formed over tensor-product or sparse grids and may be local or global, value-based or gradient-enhanced, and nodal or hierarchical. In global value-based cases (Lagrange polynomials), the barycentric formulation is used[10],[56],[49] to improve numerical efficiency and stability. Both sets of methods provide analytic response moments and variance-based metrics; however, CDF/CCDF probabilities are evaluated numerically by sampling on the expansion.
5.8. METHODS

Related Topics

5.8.7 epistemic_uncertainty_quantification_methods

Description

Epistemic uncertainty is uncertainty due to lack of knowledge.

In Dakota, epistemic uncertainty analysis is performed using interval analysis or Dempster-Shafer theory of evidence.

Note that epistemic uncertainty can also be modeled probabilistically. It would be more accurate to call this class of method, non-probabilistic uncertainty quantification, but the name persists for historical reasons.

Related Topics

- interval_estimation
- evidence_theory

Related Keywords

- global_evidence : Evidence theory with evidence measures computed with global optimization methods
- global_interval_est : Interval analysis using global optimization methods
- local_evidence : Evidence theory with evidence measures computed with local optimization methods
- local_interval_est : Interval analysis using local optimization

interval_estimation

Description

In interval analysis, one assumes that nothing is known about an epistemic uncertain variable except that its value lies somewhere within an interval. In this situation, it is NOT assumed that the value has a uniform probability of occurring within the interval. Instead, the interpretation is that any value within the interval is a possible value or a potential realization of that variable. In interval analysis, the uncertainty quantification problem is one of determining the resulting bounds on the output (defining the output interval) given interval bounds on the inputs. Again, any output response that falls within the output interval is a possible output with no frequency information assigned to it.

We have the capability to perform interval analysis using either global_interval_est or local_interval_est. In the global approach, one uses either a global optimization method or a sampling method to assess the bounds. global_interval_est allows the user to specify either lhs, which performs Latin Hypercube Sampling and takes the minimum and maximum of the samples as the bounds (no optimization is performed) or ego. In the case of ego, the efficient global optimization method is used to calculate bounds. The ego method is described in Section . If the problem is amenable to local optimization methods (e.g. can provide derivatives or use finite difference method to calculate derivatives), then one can use local methods to calculate these bounds. local_interval_est allows the user to specify either sqp which is sequential quadratic programming, or nip which is a nonlinear interior point method.

Note that when performing interval analysis, it is necessary to define interval uncertain variables as described in Section . For interval analysis, one must define only one interval per input variable, in contrast with Dempster-Shafer evidence theory, where an input can have several possible intervals. Interval analysis can be considered a
special case of Dempster-Shafer evidence theory where each input is defined by one input interval with a basic probability assignment of one. In Dakota, however, the methods are separate and semantic differences exist in the output presentation. If you are performing a pure interval analysis, we recommend using either global_interval_est or local_interval_est instead of global_evidence or local_evidence, for reasons of simplicity. An example of interval estimation is found in the Dakota/examples/users/cantilever_uq/global_interval.in, and also in Section.

Note that we have kept separate implementations of interval analysis and Dempster-Shafer evidence theory because our users often want to couple interval analysis on an outer loop” with an aleatory, probabilistic analysis on an inner loop” for nested, second-order probability calculations. See Section for additional details on these nested approaches. These interval methods can also be used as the outer loop within an interval-valued probability analysis for propagating mixed aleatory and epistemic uncertainty – refer to Section for additional details.

Interval analysis is often used to model epistemic uncertainty. In interval analysis, the uncertainty quantification problem is one of determining the resulting bounds on the output (defining the output interval) given interval bounds on the inputs.

We can do interval analysis using either global_interval_est or local_interval_est. In the global approach, one uses either a global optimization method or a sampling method to assess the bounds, whereas the local method uses gradient information in a derivative-based optimization approach.

An example of interval estimation is shown in Figure , with example results in Figure . This example is a demonstration of calculating interval bounds for three outputs of the cantilever beam problem. The cantilever beam problem is described in detail in Section . Given input intervals of [1,10] on beam width and beam thickness, we can see that the interval estimate of beam weight is approximately [1,100].

Min and Max estimated values for each response function:
weight: Min = 1.0000169352e+00 Max = 9.9999491948e+01
stress: Min = -9.7749994284e-01 Max = 2.1499428450e+01
displ: Min = -9.9315672724e-01 Max = 6.7429714485e+01

Related Topics

Related Keywords

• global_interval_est : Interval analysis using global optimization methods
• local_interval_est : Interval analysis using local optimization

evidence_theory

Description

This section discusses Dempster-Shafer evidence theory. In this approach, one does not assign a probability distribution to each uncertain input variable. Rather, one divides each uncertain input variable into one or more intervals. The input parameters are only known to occur within intervals: nothing more is assumed.

Each interval is defined by its upper and lower bounds, and a Basic Probability Assignment (BPA) associated with that interval. The BPA represents a probability of that uncertain variable being located within that interval.

The intervals and BPAs are used to construct uncertainty measures on the outputs called ”belief” and ”plausibility.” Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. For more information about the Dempster-Shafer theory of evidence, see [66] and [47].
Similar to the interval approaches, one may use global or local methods to determine plausibility and belief measures for the outputs.

Usage Notes
Note that to calculate the plausibility and belief cumulative distribution functions, one has to look at all combinations of intervals for the uncertain variables. Within each interval cell combination, the minimum and maximum value of the objective function determine the belief and plausibility, respectively. In terms of implementation, global methods use LHS sampling or global optimization to calculate the minimum and maximum values of the objective function within each interval cell, while local methods use gradient-based optimization methods to calculate these minima and maxima.

Finally, note that many non-deterministic keywords apply to the evidence methods, but one needs to be careful about the interpretation and translate probabilistic measures to epistemic ones. For example, if the user specifies distribution of type complementary, a complementary plausibility and belief function will be generated for the evidence methods (as opposed to a complementary distribution function in the sampling case). If the user specifies a set of responses levels, both the belief and plausibility will be calculated for each response level. Likewise, if the user specifies a probability level, the probability level will be interpreted both as a belief and plausibility, and response levels corresponding to the belief and plausibility levels will be calculated. Finally, if generalized reliability levels are specified, either as inputs (gen_reliability.levels) or outputs (response.levels with compute gen.reliabilities), then these are directly converted to/from probability levels and the same probability-based mappings described above are performed.

Related Topics
Related Keywords
- **global_evidence**: Evidence theory with evidence measures computed with global optimization methods
- **local_evidence**: Evidence theory with evidence measures computed with local optimization methods

5.8.8 variable support

Description
Different nondeterministic methods have differing support for uncertain variable distributions. Tables 5.37, 5.38, and 5.39 summarize the uncertain variables that are available for use by the different methods, where a "-" indicates that the distribution is not supported by the method, a "U" means the uncertain input variables of this type must be uncorrelated, a "C" denotes that correlations are supported involving uncertain input variables of this type, and an "A" means the appropriate variables must be specified as active in the variables specification block. For example, if one wants to support sampling or a stochastic expansion method over both continuous uncertain and continuous state variables, the specification **active all** must be listed in the variables specification block. Additional notes include:

- we have four variants for stochastic expansions (SE), listed as Wiener, Askey, Extended, and Piecewise which draw from different sets of basis polynomials. The term stochastic expansion indicates polynomial chaos and stochastic collocation collectively, although the Piecewise option is only currently supported for stochastic collocation. Refer to polynomial.chaos and stoch.collocation for additional information on these three options.

- methods supporting the epistemic interval distributions have differing approaches: sampling and the lhs option of global_interval_est model the interval basic probability assignments (BPAs) as continuous histogram bin distributions for purposes of generating samples; local_interval_est and the ego option of global_interval_est ignore the BPA details and models these variables as simple bounded
regions defined by the cell extremes; and \texttt{local.evidence} and \texttt{global.evidence} model the interval specifications as true BPAs.

**Related Topics**

**Related Keywords**

5.8.9 \texttt{optimization\_and\_calibration}

**Description**

Optimization algorithms work to minimize (or maximize) an objective function, typically calculated by the user simulation code, subject to constraints on design variables and responses. Available approaches in Dakota include well-tested, proven gradient-based, derivative-free local, and global methods for use in science and engineering design applications. Dakota also offers more advanced algorithms, e.g., to manage multi-objective optimization or perform surrogate-based minimization. This chapter summarizes optimization problem formulation, standard algorithms available in Dakota (mostly through included third-party libraries, see Section 6.5 of[4]), some advanced capabilities, and offers usage guidelines.

**Optimization Formulations**

This section provides a basic introduction to the mathematical formulation of optimization problems. The primary goal of this section is to introduce terms relating to these topics, and is not intended to be a description of theory or numerical algorithms. For further details, consult[8],[34],[41],[65], and [84].

A general optimization problem is formulated as follows:

\[
\begin{align*}
\text{minimize: } & \quad f(x) \\
\text{subject to: } & \quad g_L \leq g(x) \leq g_U \\
& \quad h(x) = h_t \\
& \quad a_L \leq A_i x \leq a_U \\
& \quad A_e x = a_t \\
& \quad x_L \leq x \leq x_U
\end{align*}
\]

where vector and matrix terms are marked in bold typeface. In this formulation, \( x = [x_1, x_2, \ldots, x_n] \) is an \( n \)-dimensional vector of real-valued design variables or design parameters. The \( n \)-dimensional vectors, \( x_L \) and \( x_U \), are the lower and upper bounds, respectively, on the design parameters. These bounds define the allowable values for the elements of \( x \), and the set of all allowable values is termed the design space or the parameter space. A design point or a sample point is a particular set of values within the parameter space.

The optimization goal is to minimize the objective function, \( f(x) \), while satisfying the constraints. Constraints can be categorized as either linear or nonlinear and as either inequality or equality. The nonlinear inequality constraints, \( g(x) \), are “2-sided,” in that they have both lower and upper bounds, \( g_L \) and \( g_U \), respectively. The nonlinear equality constraints, \( h(x) \), have target values specified by \( h_t \). The linear inequality constraints create a linear system \( A_i x \), where \( A_i \) is the coefficient matrix for the linear system. These constraints are also 2-sided as they have lower and upper bounds, \( a_L \) and \( a_U \), respectively. The linear equality constraints create a linear system \( A_e x \), where \( A_e \) is the coefficient matrix for the linear system and \( a_t \) are the target values. The constraints partition the parameter space into feasible and infeasible regions. A design point is said to be feasible if and only
if it satisfies all of the constraints. Correspondingly, a design point is said to be infeasible if it violates one or more of the constraints.

Many different methods exist to solve the optimization problem given in Section 6.1 of [4], all of which iterate on \( x \) in some manner. That is, an initial value for each parameter in \( x \) is chosen, the response quantities, \( f(x), g(x), h(x) \), are computed, often by running a simulation, and some algorithm is applied to generate a new \( x \) that will either reduce the objective function, reduce the amount of infeasibility, or both. To facilitate a general presentation of these methods, three criteria will be used in the following discussion to differentiate them: optimization problem type, search goal, and search method.

The optimization problem type can be characterized both by the types of constraints present in the problem and by the linearity or nonlinearity of the objective and constraint functions. For constraint categorization, a hierarchy of complexity exists for optimization algorithms, ranging from simple bound constraints, through linear constraints, to full nonlinear constraints. By the nature of this increasing complexity, optimization problem categorizations are inclusive of all constraint types up to a particular level of complexity. That is, an unconstrained problem has no constraints, a bound-constrained problem has only lower and upper bounds on the design parameters, a linearly-constrained problem has both linear and bound constraints, and a nonlinearly-constrained problem may contain the full range of nonlinear, linear, and bound constraints. If all of the linear and nonlinear constraints are equality constraints, then this is referred to as an equality-constrained problem, and if all of the linear and nonlinear constraints are inequality constraints, then this is referred to as an inequality-constrained problem. Further categorizations can be made based on the linearity of the objective and constraint functions. A problem where the objective function and all constraints are linear is called a linear programming (LP) problem. These types of problems commonly arise in scheduling, logistics, and resource allocation applications. Likewise, a problem where at least some of the objective and constraint functions are nonlinear is called a nonlinear programming (NLP) problem. These NLP problems predominate in engineering applications and are the primary focus of Dakota.

The search goal refers to the ultimate objective of the optimization algorithm, i.e., either global or local optimization. In global optimization, the goal is to find the design point that gives the lowest feasible objective function value over the entire parameter space. In contrast, in local optimization, the goal is to find a design point that is lowest relative to a “nearby” region of the parameter space. In almost all cases, global optimization will be more computationally expensive than local optimization. Thus, the user must choose an optimization algorithm with an appropriate search scope that best fits the problem goals and the computational budget.

The search method refers to the approach taken in the optimization algorithm to locate a new design point that has a lower objective function or is more feasible than the current design point. The search method can be classified as either gradient-based or nongradient-based. In a gradient-based algorithm, gradients of the response functions are computed to find the direction of improvement. Gradient-based optimization is the search method that underlies many efficient local optimization methods. However, a drawback to this approach is that gradients can be computationally expensive, inaccurate, or even nonexistent. In such situations, nongradient-based search methods may be useful. There are numerous approaches to nongradient-based optimization. Some of the more well known of these include pattern search methods (nongradient-based local techniques) and genetic algorithms (nongradient-based global techniques).

Because of the computational cost of running simulation models, surrogate-based optimization (SBO) methods are often used to reduce the number of actual simulation runs. In SBO, a surrogate or approximate model is constructed based on a limited number of simulation runs. The optimization is then performed on the surrogate model. Dakota has an extensive framework for managing a variety of local, multipoint, global, and hierarchical surrogates for use in optimization. Finally, sometimes there are multiple objectives that one may want to optimize simultaneously instead of a single scalar objective. In this case, one may employ multi-objective methods that are described in Section 6.3.1 of [4].

This overview of optimization approaches underscores that no single optimization method or algorithm works best for all types of optimization problems. Section 6.4 of [4] offers guidelines for choosing a Dakota optimization algorithm best matched to your specific optimization problem.
**Constraint Considerations** Dakota’s input commands permit the user to specify two-sided nonlinear inequality constraints of the form \( g_L \leq g_i(x) \leq g_U \), as well as nonlinear equality constraints of the form \( h_j(x) = h_t \). Some optimizers (e.g., npsol, optpp, soga, and moga methods) can handle these constraint forms directly, whereas other optimizers (e.g., async_pattern_search, dot, and conmin_mesh_adaptive_search) require Dakota to perform an internal conversion of all constraints to one-sided inequality constraints of the form \( g_i(x) \leq 0 \). In the latter case, the two-sided inequality constraints are treated as \( g_i(x) - g_U \leq 0 \) and \( g_L - g_i(x) \leq 0 \) and the equality constraints are treated as \( h_j(x) - h_t \leq 0 \) and \( h_t - h_j(x) \leq 0 \). The situation is similar for linear constraints: async_pattern_search, npsol, optpp, soga, and moga methods support them directly, whereas dot and conmin methods do not. For linear inequalities of the form \( a_L \leq a_i^T x \leq a_U \) and linear equalities of the form \( a_i^T x = a_t \), the nonlinear constraint arrays in dot and conmin methods are further augmented to include \( a_i^T x - a_U \leq 0 \) and \( a_L - a_i^T x \leq 0 \) in the inequality case and \( a_i^T x - a_t \leq 0 \) and \( a_t - a_i^T x \leq 0 \) in the equality case. Awareness of these constraint augmentation procedures can be important for understanding the diagnostic data returned from the dot and conmin methods. Other optimizers fall somewhere in between. nlpql methods support nonlinear inequality constraints \( h_j(x) = 0 \) and nonlinear one-sided inequalities \( g_i(x) \geq 0 \), but does not natively support linear constraints. Constraint mappings are used with NLPQL for both linear and nonlinear cases. Most coliny methods now support two-sided nonlinear inequality constraints and nonlinear constraints with targets, but do not natively support linear constraints.

When gradient and Hessian information is used in the optimization, derivative components are most commonly computed with respect to the active continuous variables, which in this case are the continuous design variables. This differs from parameter study methods (for which all continuous variables are active) and from nondeterministic analysis methods (for which the uncertain variables are active). Refer to Chapter 11 of[4] for additional information on derivative components and active continuous variables.

**Optimizing with Dakota: Choosing a Method**

This section summarizes the optimization methods available in Dakota. We group them according to search method and goal and establish their relevance to types of problems. For a summary of this discussion, see Section 6.4 of[4].

**Gradient-Based Local Methods** Gradient-based optimizers are best suited for efficient navigation to a local minimum in the vicinity of the initial point. They are not intended to find global optima in nonconvex design spaces. For global optimization methods, see Section 6.2.3 of[4]. Gradient-based optimization methods are highly efficient, with the best convergence rates of all of the local optimization methods, and are the methods of choice when the problem is smooth, unimodal, and well-behaved. However, these methods can be among the least robust when a problem exhibits nonsmooth, discontinuous, or multimodal behavior. The derivative-free methods described in Section 6.2.2 of[4] are more appropriate for problems with these characteristics.

Gradient accuracy is a critical factor for gradient-based optimizers, as inaccurate derivatives will often lead to failures in the search or pre-mature termination of the method. Analytic gradients and Hessians are ideal but often unavailable. If analytic gradient and Hessian information can be provided by an application code, a full Newton method will achieve quadratic convergence rates near the solution. If only gradient information is available and the Hessian information is approximated from an accumulation of gradient data, the superlinear convergence rates can be obtained. It is most often the case for engineering applications, however, that a finite difference method will be used by the optimization algorithm to estimate gradient values. Dakota allows the user to select the step size for these calculations, as well as choose between forward-difference and central-difference algorithms. The finite difference step size should be selected as small as possible, to allow for local accuracy and convergence, but not so small that the steps are “in the noise.” This requires an assessment of the local smoothness of the response functions using, for example, a parameter study method. Central differencing will generally produce more reliable gradients than forward differencing but at roughly twice the expense.
5.8. METHODS

Gradient-based methods for nonlinear optimization problems can be described as iterative processes in which a sequence of subproblems, usually which involve an approximation to the full nonlinear problem, are solved until the solution converges to a local optimum of the full problem. The optimization methods available in Dakota fall into several categories, each of which is characterized by the nature of the subproblems solved at each iteration.

Related Topics
- local_optimization_methods
- global_optimization_methods
- bayesian_calibration
- nonlinear_least_squares
- advanced_optimization

Related Keywords
- dl_solver : (Experimental) Dynamically-loaded solver

5.8.10 local_optimization_methods

Description
empty

Related Topics
- unconstrained
- constrained
- sequential_quadratic_programming

Related Keywords
- coliny_cobyla : Constrained Optimization BY Linear Approximations (COBYLA)
- nlq1_sqp : Sequential Quadratic Program
- nonlinear_cg : (Experimental) nonlinear conjugate gradient optimization
- npsol_sqp : Sequential Quadratic Program
- optpp_cg : A conjugate gradient optimization method
- optpp_fd_newton : Finite Difference Newton optimization method
- optpp_g_newton : Newton method based least-squares calibration
- optpp_newton : Newton method based optimization
- optpp_q_newton : Quasi-Newton optimization method
unconstrained

**Description**
empty

**Related Topics**

**Related Keywords**

constrained

**Description**
empty

**Related Topics**

**Related Keywords**

- coliny_cobyla: Constrained Optimization BY Linear Approximations (COBYLA)

**sequential_quadraic_programming**

**Description**
Sequential Quadratic Programming (SQP) algorithms are a class of mathematical programming problems used to solve nonlinear optimization problems with nonliner constraints. These methods are a generalization of Newton’s method: each iteration involves minimizing a quadratic model of the problem. These subproblems are formulated as minimizing a quadratic approximation of the Lagrangian subject to linearized constraints. Only gradient information is required; Hessians are approximated by low-rank updates defined by the step taken at each iteration. It is important to note that while the solution found by an SQP method will respect the constraints, the intermediate iterates may not. SQP methods available in Dakota are dot_sqp, nlpql_sqp, nlssol_sqp, and npsol_sqp. The particular implementation in nlpql_sqp uses a variant with distributed and non-monotone line search. Thus, this variant is designed to be more robust in the presence of inaccurate or noisy gradients common in many engineering applications.

**Related Topics**

**Related Keywords**

- nlpql_sqp: Sequential Quadratic Program
- nlssol_sqp: Sequential Quadratic Program for nonlinear least squares
- npsol_sqp: Sequential Quadratic Program

**5.8.11 global_optimization_methods**

**Description**
empty
5.8. METHODS

Related Topics

Related Keywords

- `asynch_pattern_search` : Pattern search, derivative free optimization method
- `coliny_direct` : DIviding RECTangles method
- `coliny_ea` : Evolutionary Algorithm
- `coliny_pattern_search` : Pattern search, derivative free optimization method
- `efficient_global` : Global Surrogate Based Optimization, a.k.a. EGO
- `ncsu_direct` : DIviding RECTangles method
- `soga` : Single-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)

5.8.12 bayesian_calibration

Description

See the discussion of Bayesian Calibration in the Dakota User’s Manual [4].

Related Topics

Related Keywords

- `bayes_calibration` : Bayesian calibration
- `dream` : DREAM (DiffeRential Evolution Adaptive Metropolis)
- `chains` : Number of chains in DREAM
- `crossover_chain_pairs` : Number of chains used in crossover.
- `gr_threshold` : Convergence tolerance for the Gelman-Rubin statistic
- `jump_step` : Number of generations a long jump step is taken
- `num_cr` : Number of candidate points for each crossover.
- `gpmsa` : (Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Markov Chain Monte Carlo algorithm with Gaussian Process Surrogate
- `adaptive_metropolis` : Use the Adaptive Metropolis MCMC algorithm
- `delayed_rejection` : Use the Delayed Rejection MCMC algorithm
- `dram` : Use the DRAM MCMC algorithm
- `metropolis_hastings` : Use the Metropolis-Hastings MCMC algorithm
- `multilevel` : Use the multilevel MCMC algorithm.
- `proposal_covariance` : Defines the technique used to generate the MCMC proposal covariance.
- `derivatives` : Use derivatives to inform the MCMC proposal covariance.
• prior: Uses the covariance of the prior distributions to define the MCMC proposal covariance.

• queso: Markov Chain Monte Carlo algorithms from the QUESO package

• adaptive_metropolis: Use the Adaptive Metropolis MCMC algorithm

• delayed_rejection: Use the Delayed Rejection MCMC algorithm

• dram: Use the DRAM MCMC algorithm

• metropolis_hastings: Use the Metropolis-Hastings MCMC algorithm

• multilevel: Use the multilevel MCMC algorithm.

• proposal_covariance: Defines the technique used to generate the MCMC proposal covariance.

• derivatives: Use derivatives to inform the MCMC proposal covariance.

• prior: Uses the covariance of the prior distributions to define the MCMC proposal covariance.

5.8.13 nonlinear_least_squares

Description

Dakota’s least squares branch currently contains three methods for solving nonlinear least squares problems:

• NL2SOL, a trust-region method that adaptively chooses between two Hessian approximations (Gauss-Newton and Gauss-Newton plus a quasi-Newton approximation to the rest of the Hessian)

• NLSSOL, a sequential quadratic programming (SQP) approach that is from the same algorithm family as NPSOL

• Gauss-Newton, which supplies the Gauss-Newton Hessian approximation to the full-Newton optimizers from OPT++.

The important difference of these algorithms from general-purpose optimization methods is that the response set is defined by calibration terms (e.g. separate terms for each residual), rather than an objective function. Thus, a finer granularity of data is used by least squares solvers as compared to that used by optimizers. This allows the exploitation of the special structure provided by a sum of squares objective function.

Related Topics

Related Keywords

• nl2sol: Trust-region method for nonlinear least squares

• nlssol_sqp: Sequential Quadratic Program for nonlinear least squares

5.8.14 advanced_optimization

Description

empty
5.9. **ADVANCED_TOPICS**

**Related Topics**
- scaling
- multiobjective_methods
- surrogate_based_optimization_methods

**Related Keywords**
scaling

**Description**
empty

**Related Topics**

**Related Keywords**
multiobjective_methods

**Description**
empty

**Related Topics**

**Related Keywords**
 surrogate_based_optimization_methods

**Description**
empty

**Related Topics**

**Related Keywords**
- efficient_global : Global Surrogate Based Optimization, a.k.a. EGO
- surrogate_based_global : Global Surrogate Based Optimization
- surrogate_based_local : Local Surrogate Based Optimization

**5.9 advanced_topics**

**Description**
Advanced Dakota capabilities
Related Topics

- advanced_strategies
- advanced_model_recursion
- advanced_simulation_interfaces
- advanced_optimization

Related Keywords

5.9.1 advanced_strategies

Description
empty

Related Topics

Related Keywords

5.9.2 advanced_model_recursion

Description
empty

Related Topics

- hybrid_and_recursions_logic

Related Keywords

hybrid_and_recursions_logic

Description
empty

Related Topics

Related Keywords

5.9.3 advanced_simulation_interfaces

Description
empty

Related Topics

- simulation_failure
- concurrency_and_parallelism
5.9. ADVANCED_TOPICS

Related Keywords

simulation_failure

Description

empty

Related Topics

Related Keywords

concurrency_and_parallelism

Description

empty

Related Topics

Related Keywords

- processors_per_analysis : Specify the number of processors per analysis when Dakota is run in parallel
- analysis_scheduling : Specify the scheduling of concurrent analyses when Dakota is run in parallel
- master : Specify a dedicated master partition for parallel analysis scheduling
- peer : Specify a peer partition for parallel analysis scheduling
- analysis_servers : Specify the number of analysis servers when Dakota is run in parallel
- asynchronous : Specify analysis driver concurrency, when Dakota is run in serial
- analysis_concurrency : Limit the number of analysis drivers within an evaluation that Dakota will schedule
- evaluation_concurrency : Determine how many concurrent evaluations Dakota will schedule
- local_evaluation_scheduling : Control how local asynchronous jobs are scheduled
- master : Specify a dedicated master partition for parallel evaluation scheduling
- peer : Specify a peer partition for parallel evaluation scheduling
- dynamic : Specify dynamic scheduling in a peer partition when Dakota is run in parallel.
- static : Specify static scheduling in a peer partition when Dakota is run in parallel.
- evaluation_servers : Specify the number of evaluation servers when Dakota is run in parallel
- processors_per_evaluation : Specify the number of processors per evaluation server when Dakota is run in parallel
- iterator_scheduling : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- master : Specify a dedicated master partition for parallel iterator scheduling
- peer : Specify a peer partition for parallel iterator scheduling
• iterator_servers : Specify the number of iterator servers when Dakota is run in parallel
• processors_per_iterator : Specify the number of processors per iterator server when Dakota is run in parallel
• iterator_scheduling : Specify the scheduling of concurrent iterators when Dakota is run in parallel
• master : Specify a dedicated master partition for parallel iterator scheduling
• peer : Specify a peer partition for parallel iterator scheduling
• iterator_servers : Specify the number of iterator servers when Dakota is run in parallel
• processors_per_iterator : Specify the number of processors per iterator server when Dakota is run in parallel
• iterator_scheduling : Specify the scheduling of concurrent iterators when Dakota is run in parallel
• master : Specify a dedicated master partition for parallel iterator scheduling
• peer : Specify a peer partition for parallel iterator scheduling
• iterator_servers : Specify the number of iterator servers when Dakota is run in parallel
• processors_per_iterator : Specify the number of processors per iterator server when Dakota is run in parallel
• iterator_scheduling : Specify the scheduling of concurrent iterators when Dakota is run in parallel
• master : Specify a dedicated master partition for parallel iterator scheduling
• peer : Specify a peer partition for parallel iterator scheduling

5.9.4 advanced_optimization

Description
empty

Related Topics
• scaling

• multiobjective_methods

• surrogate_based_optimization_methods

Related Keywords
scaling

Description
empty
5.10. PACKAGES

Related Topics
Related Keywords
multiobjective_methods

Description
easy

Related Topics
Related Keywords
surrogate_based_optimization_methods

Description
easy

5.10 packages

Description
This topic organizes information about the different software packages (libraries) that are integrated into Dakota

Related Topics

• package_coliny
• package_conmin
• package_ddace
• package_dot
• package_fsudace
• package_hopspack
• package_jega
• package_nlpql
• package_npsol
chapter 5. topics area

- package_optpp
- package_psuade
- package_queso
- package_scoli

Related Keywords
5.10.1 package_coliny

Description
SCOLIB (formerly known as COLINY) is a collection of nongradient-based optimizers that support the Common Optimization Library INterface (COLIN). SCOLIB optimizers currently include coliny_cobyla, coliny_direct, coliny_ea, coliny_pattern_search and coliny_solis_wets. (Yes, the input spec still has "coliny" prepended to the method name.) Additional SCOLIB information is available from https://software.sandia.gov/trac/acro.

SCOLIB solvers now support bound constraints and general nonlinear constraints. Supported nonlinear constraints include both equality and two-sided inequality constraints. SCOLIB solvers do not yet support linear constraints. Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds constraint_penalty times the sum of squares of the constraint violations to the objective function. Specific exceptions to this method for handling constraint violations are noted below. (The default value of constraint_penalty is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.)

The method independent controls for max_iterations and max_function_evaluations limit the number of major iterations and the number of function evaluations that can be performed during a SCOLIB optimization, respectively. The convergence_tolerance control defines the threshold value on relative change in the objective function that indicates convergence. The output verbosity specification controls the amount of information generated by SCOLIB: the silent, quiet, and normal settings correspond to minimal reporting from SCOLIB, whereas the verbose setting corresponds to a higher level of information, and debug outputs method initialization and a variety of internal SCOLIB diagnostics. The majority of SCOLIB’s methods perform independent function evaluations that can directly take advantage of Dakota’s parallel capabilities. Only coliny_solis_wets, coliny_cobyla, and certain configurations of coliny_pattern_search are inherently serial. The parallel methods automatically utilize parallel logic when the Dakota configuration supports parallelism. Lastly, neither speculative gradients nor linear constraints are currently supported with SCOLIB.

Some SCOLIB methods exploit parallelism through the use of Dakota’s concurrent function evaluations. The nature of the algorithms, however, limits the amount of concurrency that can be exploited. The maximum amount of evaluation concurrency that can be leveraged by the various methods is as follows:

- COBYLA: one
- DIRECT: twice the number of variables
- Evolutionary Algorithms: size of the population
- Pattern Search: size of the search pattern
- Solis-Wets: one
All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

Each of the SCOLIB methods supports the `solution_target` control, which defines a convergence criterion in which the optimizer will terminate if it finds an objective function value lower than the specified target.

### Related Topics

### Related Keywords

- `coliny_beta`: (Experimental) Coliny beta solver
- `coliny_cobyla`: Constrained Optimization BY Linear Approximations (COBYLA)
- `coliny_direct`: DIviding RECTangles method
- `coliny_ea`: Evolutionary Algorithm
- `coliny_pattern_search`: Pattern search, derivative free optimization method
- `coliny_solis_wets`: Simple greedy local search method

### 5.10.2 package_conmin

**Description**

The CONMIN library[83] is a public domain library of nonlinear programming optimizers, specifically the Fletcher-Reeves conjugate gradient (Dakota’s `conmin_frcg` method) method for unconstrained optimization, and the method of feasible directions (Dakota’s `conmin_mfd` method) for constrained optimization. As CONMIN was a predecessor to the DOT commercial library, the algorithm controls are very similar.

**Related Topics**

**Related Keywords**

- `conmin`: Access to methods in the CONMIN library
- `frcg`: A conjugate gradient optimization method
- `mfd`: Method of feasible directions

### 5.10.3 package_ddace

**Description**

The Distributed Design and Analysis of Computer Experiments (DDACE) library provides the following DA-CE techniques: grid sampling (`grid`), pure random sampling (`random`), orthogonal array sampling (`oas`),
latin hypercube sampling (lhs), orthogonal array latin hypercube sampling (oa_lhs), Box-Behnken (box_behnken), and central composite design (central_composite).

It is worth noting that there is some overlap in sampling techniques with those available from the nondeterministic branch. The current distinction is that the nondeterministic branch methods are designed to sample within a variety of probability distributions for uncertain variables, whereas the design of experiments methods treat all variables as having uniform distributions. As such, the design of experiments methods are well-suited for performing parametric studies and for generating data sets used in building global approximations, but are not currently suited for assessing the effect of uncertainties characterized with probability distribution. If a design of experiments over both design/state variables (treated as uniform) and uncertain variables (with probability distributions) is desired, then sampling can support this with active all specified in the Variables specification block.

Related Topics

Related Keywords

• dace : Design and Analysis of Computer Experiments

5.10.4 package_dot

Description
The DOT library [85] contains nonlinear programming optimizers, specifically the Broyden-Fletcher-Goldfarb-Shanno (Dakota’s dot_bfgs method) and Fletcher-Reeves conjugate gradient (Dakota’s dot_frcg method) methods for unconstrained optimization, and the modified method of feasible directions (Dakota’s dot_mmfd method), sequential linear programming (Dakota’s dot_slp method), and sequential quadratic programming (Dakota’s dot_sqp method) methods for constrained optimization.

Related Topics

Related Keywords

• dot : Access to methods in the DOT package
• bfgs : A conjugate gradient optimization method
• frcg : A conjugate gradient optimization method
• mmfd : Method of feasible directions
• slp : Sequential Linear Programming
• sqp : Sequential Quadratic Program

5.10.5 package_fsudoace

Description
The Florida State University Design and Analysis of Computer Experiments (FSUDace) library provides the following DACE techniques: quasi-Monte Carlo sampling (fsu_quasi_mc) based on the Halton sequence (halton) or the Hammersley sequence (hammersley), and Centroidal Voronoi Tessellation (fsu_cvt).
5.10. PACKAGES

Related Topics

Related Keywords

- **quality_metrics** : Calculate metrics to assess the quality of quasi-Monte Carlo samples
- **fsu_cvt** : Design of Computer Experiments - Centroidal Voronoi Tessellation
- **quality_metrics** : Calculate metrics to assess the quality of quasi-Monte Carlo samples
- **halton** : Generate samples from a Halton sequence
- **fsu_quasi_mc** : Design of Computer Experiments - Quasi-Monte Carlo sampling
- **halton** : Generate samples from a Halton sequence
- **hammersley** : Use Hammersley sequences
- **quality_metrics** : Calculate metrics to assess the quality of quasi-Monte Carlo samples

5.10.6 package hopspack

Description

The HOPSPACK software \[69\] contains the asynchronous parallel pattern search (APPS) algorithm \[37\]. It can handle unconstrained problems as well as those with bound constraints, linear constraints, and general nonlinear constraints.

HOPSPACK is available to the public under the GNU LGPL and the source code is included with Dakota. HOPSPACK-specific software documentation is available from [https://software.sandia.gov/trac/hopspack](https://software.sandia.gov/trac/hopspack).

Related Topics

Related Keywords

- **asynch_pattern_search** : Pattern search, derivative free optimization method

5.10.7 package jega

Description

The JEGA library[19] contains two global optimization methods. The first is a Multi-objective Genetic Algorithm (MOGA) which performs Pareto optimization. The second is a Single-objective Genetic Algorithm (SOGA) which performs optimization on a single objective function. Both methods support general constraints and a mixture of real and discrete variables. The JEGA library was written by John Eddy, currently a member of the technical staff in the System Readiness and Sustainment Technologies department at Sandia National Laboratories in Albuquerque. These algorithms are accessed as moga and soga within Dakota.

Related Topics

Related Keywords

- **moga** : Multi-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)
- **soga** : Single-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)
5.10.8 package nlpql

Description

The NLPQL library is a commercially-licensed library containing a sequential quadratic programming (SQP) optimizer, specified as Dakota’s nlpql_sqp method, for constrained optimization. The particular implementation used is NLPQLP [74], a variant with distributed and non-monotone line search.

Related Topics

Related Keywords

• nlpql_sqp : Sequential Quadratic Program

5.10.9 package npsol

Description

The NPSOL library[33] contains a sequential quadratic programming (SQP) implementation (the npsol_sqp method). SQP is a nonlinear programming optimizer for constrained minimization.

Related Topics

Related Keywords

• npsol_sqp : Sequential Quadratic Program

5.10.10 package optpp

Description

The OPT++ library[60] contains primarily gradient-based nonlinear programming optimizers for unconstrained, bound-constrained, and nonlinearly constrained minimization: Polak-Ribiere conjugate gradient (Dakota’s optpp-cg method), quasi-Newton (Dakota’s optpp_q_newton method), finite difference Newton (Dakota’s optpp_fd_newton method), and full Newton (Dakota’s optpp_newton method).

The conjugate gradient method is strictly unconstrained, and each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a nonlinear interior-point approach to manage the constraints. The library also contains a direct search algorithm, PDS (parallel direct search, Dakota’s optpp_pds method), which supports bound constraints.

Controls

1. max_iterations
2. max_function_evaluations
3. convergence_tolerance
4. output
5. speculative
Concurrency
OPT++’s gradient-based methods are not parallel algorithms and cannot directly take advantage of concurrent function evaluations. However, if numerical_gradients with method_source dakota is specified, a parallel Dakota configuration can utilize concurrent evaluations for the finite difference gradient computations.

Constraints
Linear constraint specifications are supported by each of the Newton methods (optpp_newton, optpp_q_newton, optpp_fd_newton, and optpp_g_newton)
* optpp_cg must be unconstrained
* optpp_pds can be, at most, bound-constrained.

Related Topics

**Related Keywords**
- **optpp_cg**: A conjugate gradient optimization method
- **optpp_fd_newton**: Finite Difference Newton optimization method
- **optpp_g_newton**: Newton method based least-squares calibration
- **optpp_newton**: Newton method based optimization
- **optpp_pds**: Simplex-based derivative free optimization method
- **optpp_q_newton**: Quasi-Newton optimization method

### 5.10.11 package_psuade

**Description**
The Problem Solving Environment for Uncertainty Analysis and Design Exploration (PSUADE) is a Lawrence Livermore National Laboratory tool for metamodeling, sensitivity analysis, uncertainty quantification, and optimization. Its features include non-intrusive and parallel function evaluations, sampling and analysis methods, an integrated design and analysis framework, global optimization, numerical integration, response surfaces (MARS and higher order regressions), graphical output with Pgplot or Matlab, and fault tolerance [81].

**Related Topics**

**Related Keywords**
- **psuade_moat**: Morris One-at-a-Time

### 5.10.12 package_queso

**Description**
QUESO stands for Quantification of Uncertainty for Estimation, Simulation, and Optimization. It supports Bayesian calibration methods. It is developed at The University of Texas at Austin.
CHAPTER 5. TOPICS AREA

Related Topics

Related Keywords

- bayes_calibration: Bayesian calibration
- gpmsa: (Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Markov Chain Monte Carlo algorithm with Gaussian Process Surrogate
- queso: Markov Chain Monte Carlo algorithms from the QUESO package

5.10.13 package_scolib

Description

SCOLIB (formerly known as COLINY) is a collection of nongradient-based optimizers that support the Common Optimization Library INterface (COLIN). SCOLIB optimizers currently include coliny_cobyla, coliny_direct, coliny_ea, coliny_pattern_search and coliny_solis_wets. (Yes, the input spec still has "coliny" prepended to the method name.) Additional SCOLIB information is available from https://software.sandia.gov/trac/acro.

SCOLIB solvers now support bound constraints and general nonlinear constraints. Supported nonlinear constraints include both equality and two-sided inequality constraints. SCOLIB solvers do not yet support linear constraints. Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds $constraint\_penalty \times \sum \text{of constraint violations}$ to the objective function. Specific exceptions to this method for handling constraint violations are noted below. (The default value of $constraint\_penalty$ is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.)

The method independent controls for $max\_iterations$ and $max\_function\_evaluations$ limit the number of major iterations and the number of function evaluations that can be performed during a SCOLIB optimization, respectively. The $convergence\_tolerance$ control defines the threshold value on relative change in the objective function that indicates convergence. The $output\_verbosity$ specification controls the amount of information generated by SCOLIB: the silent, quiet, and normal settings correspond to minimal reporting from SCOLIB, whereas the verbose setting corresponds to a higher level of information, and debug outputs method initialization and a variety of internal SCOLIB diagnostics. The majority of SCOLIB’s methods perform independent function evaluations that can directly take advantage of Dakota’s parallel capabilities. Only coliny_solis_wets, coliny_cobyla, and certain configurations of coliny_pattern_search are inherently serial. The parallel methods automatically utilize parallel logic when the Dakota configuration supports parallelism. Lastly, neither speculative gradients nor linear constraints are currently supported with SCOLIB.

Some SCOLIB methods exploit parallelism through the use of Dakota’s concurrent function evaluations. The nature of the algorithms, however, limits the amount of concurrency that can be exploited. The maximum amount of evaluation concurrency that can be leveraged by the various methods is as follows:

- COBYLA: one
- DIRECT: twice the number of variables
- Evolutionary Algorithms: size of the population
- Pattern Search: size of the search pattern
- Solis-Wets: one
All SCOLIB methods support the `show_mispec_opt` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_opt` option provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

Each of the SCOLIB methods supports the `solution_target` control, which defines a convergence criterion in which the optimizer will terminate if it finds an objective function value lower than the specified target.

**Related Topics**

**Related Keywords**

- `coliny_beta` : (Experimental) Coliny beta solver
- `coliny_cobyla` : Constrained Optimization BY Linear Approximations (COBYLA)
- `coliny_direct` : DIviding RECTangles method
- `coliny_ea` : Evolutionary Algorithm
- `coliny_pattern_search` : Pattern search, derivative free optimization method
- `coliny_solis_wets` : Simple greedy local search method
<table>
<thead>
<tr>
<th>Distribution Type</th>
<th>Sampling</th>
<th>Local Reliability</th>
<th>Global Reliability</th>
<th>Wiener SE</th>
<th>Askey SE</th>
<th>Extended SE</th>
<th>Piecewise SE</th>
<th>Local Interval</th>
<th>Global Interval</th>
<th>Local Evidence</th>
<th>Global Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Bounded Normal</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Log-normal</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Bounded Log-normal</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Uniform</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Log-uniform</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Triangular</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Exponential</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Beta</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gamma</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gumbel</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Frechet</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Weibull</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Continuous Histogram Bin</td>
<td>C</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.1: Summary of Distribution Types supported by Nondeterministic Methods, Part I (Continuous Aleatory Types)
### Table 5.2: Summary of Distribution Types supported by Nondeterministic Methods, Part II (Discrete Aleatory Types)

<table>
<thead>
<tr>
<th>Distribution Type</th>
<th>Sampling</th>
<th>Local Reliability</th>
<th>Global Reliability</th>
<th>Wiener SE</th>
<th>Askey SE</th>
<th>Extended SE</th>
<th>Piecewise SE</th>
<th>Local Interval</th>
<th>Global Interval</th>
<th>Local Evidence</th>
<th>Global Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poisson</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Binomial</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Negative Binomial</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Geometric</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Discrete Histogram Point</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Distribution Type</td>
<td>Sampling</td>
<td>Local Reliability</td>
<td>Global Reliability</td>
<td>Wiener SE</td>
<td>Askey SE</td>
<td>Extended SE</td>
<td>Piecewise SE</td>
<td>Local Interval</td>
<td>Global Interval</td>
<td>Local Evidence</td>
<td>Global Evidence</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>-------------------</td>
<td>--------------------</td>
<td>-----------</td>
<td>----------</td>
<td>------------</td>
<td>-------------</td>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Interval</td>
<td>U</td>
<td>-</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>Continuous Design</td>
<td>U,A</td>
<td>-</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Discrete Design</td>
<td>U,A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Range, Int Set, Real Set</td>
<td>U,A</td>
<td>-</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Continuous State</td>
<td>U,A</td>
<td>-</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>U,A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Discrete State</td>
<td>U,A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Range, Int Set, Real Set</td>
<td>U,A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.3: Summary of Distribution Types supported by Nondeterministic Methods, Part III (Epistemic, Design, and State Types)
Chapter 6

Keywords Area

This page lists the six blocks. From here, you can navigate to every keyword.

- environment
- method
- model
- variables
- interface
- responses

Introduction to Dakota Keywords

In Dakota, the environment manages execution modes and I/O streams and defines the top-level iterator. Generally speaking, an iterator contains a model and a model contains a set of variables, an interface, and a set of responses. An iterator repeatedly operates on the model to map the variables into responses using the interface. Each of these six components (environment, method, model, variables, interface, and responses) are separate specifications in the user’s input file, and as a whole, determine the study to be performed during an execution of the Dakota software.

A Dakota execution is limited to a single environment, but may involve multiple methods and multiple models. In particular, advanced iterators (i.e., meta- and component-based iterators) and advanced models (i.e., nested and surrogate models) may specialize to include recursions with additional sub-iterators and sub-models. Since each model may contain its own variables, interface, and responses, there may be multiple specifications of the method, model, variables, interface, and responses sections.

Keyword Pages

Every Dakota keyword has its own page in this manual. The page describes:

- Whether the keyword takes ARGUMENTS, and the data type Additional notes about ARGUMENTS can be found here: Specifying Arguments.
- Whether it has an ALIAS
- Which additional keywords can be specified to change its behavior
- Which of these additional keywords are required or optional
- Additional information about how to use the keyword in an input file
6.1 environment

- Keywords Area
- environment

Top-level settings for Dakota execution

Topics

This keyword is related to the topics:

- block

Specification

Alias: none  
Argument(s): none  
Default: no environment

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>check</td>
<td></td>
<td>Invoke Dakota in input check mode</td>
</tr>
<tr>
<td>Optional</td>
<td>output_file</td>
<td></td>
<td>Base filename for output redirection</td>
</tr>
<tr>
<td>Optional</td>
<td>error_file</td>
<td></td>
<td>Base filename for error redirection</td>
</tr>
<tr>
<td>Optional</td>
<td>read_restart</td>
<td></td>
<td>Base filename for restart file read</td>
</tr>
<tr>
<td>Optional</td>
<td>write_restart</td>
<td></td>
<td>Base filename for restart file write</td>
</tr>
<tr>
<td>Optional</td>
<td>pre_run</td>
<td></td>
<td>Invoke Dakota with pre-run mode active</td>
</tr>
<tr>
<td>Optional</td>
<td>run</td>
<td></td>
<td>Invoke Dakota with run mode active</td>
</tr>
</tbody>
</table>
6.1. ENVIRONMENT

Optional  

| post_run | Invoke Dakota with post-run mode active |

Optional  

| graphics | Display a 2D graphics window of variables and responses |

Optional  

| tabular_data | Write a tabular results file with variable and response history |

Optional  

| output_precision | Control the output precision |

Optional  

| results_output | (Experimental) Write a summary file containing the final results |

Optional  

| top_method_-pointer | Identify which method leads the Dakota study |

**Description**

The environment section in a Dakota input file is optional. It specifies the top-level solution environment, optionally including run modes, output controls, and identification of the primary iterative method (`top_method_-pointer`). The output-related keywords address graphics, generation of tabular and results data, and precision of numerical output.

**Run Mode Defaults**

Dakota run phases include `check`, `pre_run`, `run`, and `post_run`. The default behavior is to `pre_run`, `run`, and `post_run`, though any or all of these may be specified to select specific run phases. Specifying `check` will cause Dakota to exit before any selected run modes.

**6.1.1 check**

- **Keywords Area**

- **environment**

- **check**

  Invoke Dakota in input check mode

**Topics**

This keyword is related to the topics:

- `command_line_options`
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none
Default: no check; proceed to run

Description

When specified, Dakota input will be parsed and the problem instantiated. Dakota will exit reporting whether any errors were found.

6.1.2 output_file

- Keywords Area
- environment
- output_file

Base filename for output redirection

Topics

This keyword is related to the topics:

- dakota_IO
- command_line_options

Specification

Alias: none
Argument(s): STRING
Default: output to console, not file

Description

Specify a base filename to which Dakota output will be directed. Output will (necessarily) be redirected after the input file is parsed. This option is overridden by any command-line -output option.

Default Behavior
Output to console (screen).

6.1.3 error_file

- Keywords Area
- environment
- error_file

Base filename for error redirection
6.1. ENVIRONMENT

Topics
This keyword is related to the topics:

- dakota.IO
- command_line_options

Specification
Alias: none

Argument(s): STRING
Default: errors to console, not file

Description
Specify a base filename to which Dakota errors will be directed. Errors will (necessarily) be redirected after the input file is parsed. This option is overridden by any command-line -error option.

Default Behavior
Errors to console (screen).

6.1.4 read_restart

Keywords Area
- environment
- read_restart

Base filename for restart file read

Topics
This keyword is related to the topics:

- dakota.IO
- command_line_options

Specification
Alias: none

Argument(s): STRING
Default: no restart read

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>stop_restart</td>
</tr>
</tbody>
</table>

Evaluation ID number at which to stop reading restart file
CHAPTER 6. KEYWORDS AREA

Description

Specify a base filename for the restart file Dakota should read. This option is overridden by any command-line -read_restart option.

Default Behavior

No restart file is read.

stop_restart

- Keywords Area
- environment
- read_restart
- stop_restart

Evaluation ID number at which to stop reading restart file

Topics

This keyword is related to the topics:

- dakota_IO

Specification

Alias: none

Argument(s): INTEGER

Default: read all records

Description

This option is overridden by any command-line -stop_restart option.

6.1.5 write_restart

- Keywords Area
- environment
- write_restart

Base filename for restart file write

Topics

This keyword is related to the topics:

- dakota_IO
- command_line_options
6.1. **ENVIRONMENT**

**Specification**

*Alias:* none

*Argument(s):* STRING

*Default:* dakota.rst

**Description**

Specify a base filename for the restart file Dakota should write. This option is overridden by any command-line -write_restart option.

### 6.1.6 pre_run

- **Keywords Area**
  - environment
  - pre_run

Invoke Dakota with pre-run mode active

**Topics**

This keyword is related to the topics:

- command_line_options

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* pre-run, run, post-run all executed

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>input</td>
<td></td>
<td>Base filename for pre-run mode data input</td>
</tr>
<tr>
<td>Optional</td>
<td>output</td>
<td></td>
<td>Base filename for pre-run mode data output</td>
</tr>
</tbody>
</table>

**Description**

When specified, Dakota execution will include the pre-run mode, which sets up methods and often generates parameter sets to evaluate. This mode is currently useful for parameter study, DACE, and Monte Carlo sampling methods.

**Default Behavior**

When no run modes are specified, Dakota will perform pre-run, run, and post-run phases.
CHAPTER 6. KEYWORDS AREA

input

- Keywords Area
- environment
- pre_run
- input

Base filename for pre-run mode data input

Topics

This keyword is related to the topics:

- dakota_IO

Specification

Alias: none
Argument(s): STRING
Default: no pre-run specific input read

Description

(For future expansion; not currently used by any methods.) Specify a base filename from which Dakota will read any pre-run input data. This option is overridden by any command-line -pre_run arguments.

output

- Keywords Area
- environment
- pre_run
- output

Base filename for pre-run mode data output

Topics

This keyword is related to the topics:

- dakota_IO

Specification

Alias: none
Argument(s): STRING
Default: no pre-run specific output written
### Description

Specify a base filename to which Dakota will write any pre-run output data (typically parameter sets to be evaluated). This option is overridden by any command-line -pre_run arguments.

**Usage Tips**

Dakota exports tabular data in one of three formats:

- **annotated** (default)
- **custom.annotated**
- **freeform**

**annotated**

- **Keywords Area**
- **environment**
- **pre_run**
- **output**
- **annotated**

Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

- **file_formats**

**Specification**

 Alias: none

 Argument(s): none

 Default: annotated format
CHAPTER 6. KEYWORDS AREA

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify `custom_annotated header eval id`.
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated

Resulting tabular file:
% eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

custom_annotated

- **Keywords Area**
- **environment**
- **pre_run**
- **output**
- **custom_annotated**

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- **file formats**
6.1. ENVIRONMENT

Specification

Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>header</td>
<td>eval_id</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom.annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom.annotated header eval_id
```

Resulting tabular file:
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
</tbody>
</table>

header

- Keywords Area
- environment
- pre_run
- output
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

Alias: none

Argument(s): none

Default: no header

**Description**

See description of parent custom_annotated

**eval_id**

- Keywords Area
- environment
- pre_run
- output
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

Alias: none

Argument(s): none

Default: no eval_id column
6.1. ENVIRONMENT

Description
See description of parent custom_annotated

interface_id
  - Keywords Area
  - environment
  - pre_run
  - output
  - custom_annotated
  - interface_id

Enable interface ID column in custom-annotated tabular file

Specification
Alias: none
  Argument(s): none
  Default: no interface_id column

Description
See description of parent custom_annotated

freeform
  - Keywords Area
  - environment
  - pre_run
  - output
  - freeform

Selects freeform file format

Topics
This keyword is related to the topics:
  - file_formats

Specification
Alias: none
  Argument(s): none
  Default: annotated format
CHAPTER 6. KEYWORDS AREA

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
• In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
 0.9  1.1  0.0002  0.26  0.76
 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

6.1.7 run

• Keywords Area
• environment
• run

Invoke Dakota with run mode active

Topics

This keyword is related to the topics:

• command_line_options

Specification

Alias: none

Argument(s): none

Default: pre-run, run, post-run all executed
### Description

When specified, Dakota execution will include the run mode, which invokes interfaces to map parameters to responses.

**Default Behavior**

When no run modes are specified, Dakota will perform pre-run, run, and post-run phases.

#### input

- **Keywords Area**
- **environment**
- **run**
- **input**

Base filename for run mode data input

### Topics

This keyword is related to the topics:

- *dakota*\_IO

### Specification

**Alias:** none

**Argument(s):** STRING

**Default:** no run specific input read

(For future expansion; not currently used by any methods.) Specify a base filename from which Dakota will read any run input data, such as parameter sets to evaluate. This option is overridden by any command-line -run arguments.
output
  - Keywords Area
  - environment
  - run
  - output

Base filename for run mode data output

Topics
This keyword is related to the topics:
  - dakota IO

Specification
Alias: none
Argument(s): STRING
Default: no run specific output written

Description
(For future expansion; not currently used by any methods.) Specify a base filename to which Dakota will write any run output data (typically parameter, response pairs). This option is overridden by any command-line -run arguments.

6.1.8 post_run
  - Keywords Area
  - environment
  - post_run

Invoke Dakota with post-run mode active

Topics
This keyword is related to the topics:
  - command_line_options

Specification
Alias: none
Argument(s): none
Default: pre-run, run, post-run all executed
### Description

When specified, Dakota execution will include the post-run mode, which analyzes parameter/response data sets and computes final results. This mode is currently useful for parameter study, DACE, and Monte Carlo sampling methods.

**Default Behavior**

When no run modes are specified, Dakota will perform pre-run, run, and post-run phases.

**input**

- Keywords Area
- environment
- post_run
- input

Base filename for post-run mode data input

**Topics**

This keyword is related to the topics:

- Dakota.IO

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** no post-run specific input read

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>Dakota Keyword</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>Description</td>
</tr>
</tbody>
</table>

Base filename for post-run mode data input

Selects annotated tabular file format
**Description**

Specify a base filename from which Dakota will read any post-run input data, such as parameter/response data on which to calculate final statistics. This option is overridden by any command-line -post_run arguments.

**Usage Tips**

Dakota imports tabular data in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**annotated**

- Keywords Area
- environment
- post_run
- input
- annotated

Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none

Argument(s): none

Default: annotated format
6.1. ENVIRONMENT

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

• To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annnotated header eval_id

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id    interface     x1    x2    obj_fn    nln_ineq_con_1  nln_ineq_con_2
1           NO_ID        0.9    1.1    0.0002    0.26            0.76
2           NO_ID        0.90009 1.1    0.0001996404857 0.2601620081 0.759955
3           NO_ID        0.89991 1.1    0.0002003604863 0.2598380081 0.760045
...```

custom_annnotated

• Keywords Area

• environment

• post_run

• input

• custom_annnotated

  Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

• file_formats
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: annotated format

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:
6.1. ENVIRONMENT

%eval_id  x1   x2   obj_fn nln_ineq_con_1 nln_ineq_con_2
1  0.9   1.1   0.0002  0.26  0.76
2  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...

header
- Keywords Area
- environment
- post_run
- input
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent `custom_annotated`

**eval_id**

- Keywords Area
- environment
- post_run
- input
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column
CHAPTER 6. KEYWORDS AREA

Description
See description of parent custom_annotated

interface_id
- Keywords Area
- environment
- post_run
- input
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no interface_id column

Description
See description of parent custom_annotated

freeform
- Keywords Area
- environment
- post_run
- input
- freeform

Selects freeform file format

Topics
This keyword is related to the topics:
- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format
6.1. ENVIRONMENT

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```dakota
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
0.9  1.1  0.0002  0.26  0.76
0.90009 1.1  0.0001996404857  0.2601620081  0.759955
0.89991 1.1  0.0002003604863  0.2598380081  0.760045
...
```

output
- Keywords Area
- environment
- post_run
- output

Base filename for post-run mode data output

Topics
This keyword is related to the topics:
- dakota, IO
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: no post-run specific output written

Description

(For future expansion; not currently used by any methods.) Specify a base filename to which Dakota will write any post-run output data. This option is overridden by any command-line -post_run arguments.

6.1.9 graphics

- Keywords Area

- environment

- graphics

Display a 2D graphics window of variables and responses

Topics

This keyword is related to the topics:

- dakota_output

Specification

Alias: none

Argument(s): none

Default: graphics off

Description

For most studies, the graphics flag activates a 2D graphics window containing history plots for the variables and response functions in the study. This window is updated in an event loop with approximately a 2 second cycle time. Some study types such as surrogate-based optimization or local reliability specialize the use of the graphics window.

There is no dependence between the graphics flag and the tabular_data flag; they may be used independently or concurrently.

See Also

These keywords may also be of interest:

- tabular_data
6.1.10 *tabular_data*

- Keywords Area
  - environment
  - *tabular_data*

Write a tabular results file with variable and response history

**Topics**

This keyword is related to the topics:

- *dakota_output*

**Specification**

**Alias:** *tabular_graphics_data*

**Argument(s):** none

**Default:** no tabular data output

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><em>tabular_data_file</em></td>
<td>File name for tabular data output</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td><em>custom_annotated</em></td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>freeform</em></td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

**Description**

Specifying the *tabular_data* flag writes to a data file the same variable and response function history data plotted when using the *graphics* flag. Within the generated data file, the variables and response functions appear as columns and each function evaluation provides a new table row. This capability is most useful for post-processing of Dakota results with third-party graphics tools such as MatLab, Excel, Tecplot, etc.

There is no dependence between the *graphics* flag and the *tabular_data* flag; they may be used independently or concurrently.

Dakota exports tabular data in one of three formats:

- *annotated* (default)
- *custom_annotated*
- *freeform*
See Also

These keywords may also be of interest:

- graphics
- tabular
- file
- Keywords Area
- environment
- tabular_data
- tabular_data_file

File name for tabular data output

Topics

This keyword is related to the topics:

- dakota_output

Specification

Alias: tabular_graphics_file
Argument(s): STRING
Default: dakota_tabular.dat

Description

Specifies a name to use for the tabular data file, overriding the default dakota_tabular.dat.

annotated

- Keywords Area
- environment
- tabular_data
- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats
6.1. ENVIRONMENT

Specification

Alias: none
Argument(s): none
Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

• To specify pre-Dakota 6.1 tabular format, which did not include interface id, specify custom_annotated header eval_id

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

custom_annotated

• Keywords Area

• environment

• tabular_data

• custom_annotated

Selects custom-annotated tabular file format
Topics

This keyword is related to the topics:

- file formats

Specification

Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:
### 6.1. ENVIRONMENT

```python
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id

Resulting tabular file:

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
</tbody>
</table>
```

**header**

- **Keywords Area**
- **environment**
- **tabular_data**
- **custom_annotated**
- **header**

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no header

**Description**

See description of parent **custom_annotated**

**eval_id**

- **Keywords Area**
- **environment**
- **tabular_data**
- **custom_annotated**
- **eval_id**

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no eval_id column
Description
See description of parent custom_annotated

interface_id

- Keywords Area
- environment
- tabular_data
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no interface_id column

Description
See description of parent custom_annotated

freeform

- Keywords Area
- environment
- tabular_data
- freeform

Selects freeform file format

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format
6.1. ENVIRONMENT

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
• In freeform, the num_rows x numCols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
0.9  1.1  0.0002  0.26  0.76
0.90009  1.1  0.0001996404857  0.2601620081  0.759955
0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

6.1.11 output_precision

• Keywords Area
  • environment
  • output_precision

Control the output precision

Topics
This keyword is related to the topics:
• dakota_output
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGER
Default: 10

Description

The precision of numeric output precision can be set with \texttt{output\_precision}, with an upper limit of 16. When not specified, most Dakota output will default to a precision of 10, though filesystem interfaces and pre-run output use higher precision for accuracy and better results reproducibility.

6.1.12 results\_output

- Keywords Area
- environment
- results\_output

(Experimental) Write a summary file containing the final results

Topics

This keyword is related to the topics:

- dakota\_output

Specification

Alias: none

Argument(s): none
Default: no results output

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>results_output_file</td>
<td></td>
<td>The base file name of the results file</td>
</tr>
</tbody>
</table>

Description

Final results from a Dakota study can be output to \texttt{dakota\_results\_txt} by specifying \texttt{results\_output} (optionally specifying an alternate file name with \texttt{results\_output\_filename}). The current experimental text file format is hierarchical and a precursor to planned output to structured text formats such as XML or YAML, and binary formats such as HDF5. The contents, organization, and format of results files are all under active development and are subject to change.
6.1. ENVIRONMENT

results_output_file

- Keywords Area
- environment
- results_output
- results_output_file

The base file name of the results file

Topics

This keyword is related to the topics:

- dakota_output

Specification

Alias: none

Argument(s): STRING

Default: dakota_results.txt

Description

Default file name is dakota_results.txt

6.1.13 top_method_pointer

- Keywords Area
- environment
- top_method_pointer

Identify which method leads the Dakota study

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: method_pointer

Argument(s): STRING

Default: see discussion
CHAPTER 6. KEYWORDS AREA

Description

An optional top_method_pointer specification may be used to point to a particular method specification that will lead the Dakota analysis. The associated string must be a method identifier specified via id_method. If top_method_pointer is not used, then it will be inferred as described below (no top_method_pointer within an environment specification is treated the same as no environment specification).

Default Behavior

The top_method_pointer keyword is typically used in Dakota studies consisting of more than one method block to clearly indicate which is the leading method. This method provides the starting point for the iteration. The corresponding method specification may recurse with additional sub-method pointers in the case of "meta-iteration" (see method) or may specify a single method without recursion. Either case will ultimately result in identification of one or more model specifications using model_pointer, which again may or may not involve further recursion (see nested and surrogate for recursion cases). Each of the model specifications identify the variables and responses specifications (using variables_pointer and responses_pointer) that are used to build the model, and depending on the type of model, may also identify an interface specification (for example, using interface_pointer). If one of these specifications does not provide an optional pointer, then that component will be constructed using the last specification parsed.

When the environment block is omitted, the top level method will be inferred as follows: When a single method is specified, there is no ambiguity and the sole method will be the top method. When multiple methods are specified, the top level method will be deduced from the hierarchical relationships implied by method pointers. If this inference is not well defined (e.g., there are multiple method specifications without any pointer relationship), then the default behavior is to employ the last method specification parsed.

Examples

Specify that the optimization method is the outermost method in an optimization under uncertainty study

```
environment
  top_method_pointer 'OPTIMIZATION_METHOD'
method
  id_method 'UQ_METHOD'
  ...
method
  id_method 'OPTIMIZATION_METHOD'
  ...
```

See Also

These keywords may also be of interest:

- id_method

6.2 method

- Keywords Area
- method

Begins Dakota method selection and behavioral settings.

Topics

This keyword is related to the topics:

- block
6.2. METHOD

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>id_method</td>
<td>Name the method block; helpful when there are multiple</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>output</td>
<td>Control how much method information is written to the screen and output file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>final_solutions</td>
<td>Number of designs returned as the best solutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hybrid</td>
<td>Strategy in which a set of methods synergistically seek an optimal design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multi_start</td>
<td>Multi-Start Optimization Method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pareto_set</td>
<td>Pareto set optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>branch_and_bound</td>
<td>(Experimental Capability) Solves a mixed integer nonlinear optimization problem</td>
</tr>
<tr>
<td>surrogate_based_local</td>
<td>Local Surrogate Based Optimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>surrogate_based_global</td>
<td>Global Surrogate Based Optimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dot_frcg</td>
<td>A conjugate gradient optimization method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dot_mmfd</td>
<td>Method of feasible directions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dot_bfgs</td>
<td>A conjugate gradient optimization method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dot_slp</td>
<td>Sequential Linear Programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dot_sqp</td>
<td>Sequential Quadratic Program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dot</td>
<td>Access to methods in the DOT package</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conmin_frcg</td>
<td>A conjugate gradient optimization method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conmin_mfd</td>
<td>Method of feasible directions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conmin</td>
<td>Access to methods in the CONMIN library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dl_solver</td>
<td>(Experimental) Dynamically-loaded solver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>npsol_sqp</td>
<td>Sequential Quadratic Program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nlssol_sqp</td>
<td>Sequential Quadratic Program for nonlinear least squares</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stanford</td>
<td>Select methods from the Stanford package</td>
</tr>
<tr>
<td>nlpql.sqp</td>
<td>Sequential Quadratic Program</td>
</tr>
<tr>
<td>optpp.cg</td>
<td>A conjugate gradient optimization method</td>
</tr>
<tr>
<td>optpp.q.newton</td>
<td>Quasi-Newton optimization method</td>
</tr>
<tr>
<td>optpp.fd.newton</td>
<td>Finite Difference Newton optimization method</td>
</tr>
<tr>
<td>optpp.g.newton</td>
<td>Newton method based least-squares calibration</td>
</tr>
<tr>
<td>optpp.newton</td>
<td>Newton method based optimization</td>
</tr>
<tr>
<td>optpp.pds</td>
<td>Simplex-based derivative free optimization method</td>
</tr>
<tr>
<td>asynch_pattern-search</td>
<td>Pattern search, derivative free optimization method</td>
</tr>
<tr>
<td>mesh_adaptive-search</td>
<td>Finds optimal variable values using adaptive mesh-based search</td>
</tr>
<tr>
<td>moga</td>
<td>Multi-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)</td>
</tr>
<tr>
<td>soga</td>
<td>Single-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)</td>
</tr>
<tr>
<td>keyword</td>
<td>description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>coliny_pattern_search</td>
<td>Pattern search, derivative free optimization method</td>
</tr>
<tr>
<td>coliny_solis_wets</td>
<td>Simple greedy local search method</td>
</tr>
<tr>
<td>coliny_cobyla</td>
<td>Constrained Optimization BY Linear Approximations (COBYLA)</td>
</tr>
<tr>
<td>coliny_direct</td>
<td>DIviding RECTangles method</td>
</tr>
<tr>
<td>coliny_ea</td>
<td>Evolutionary Algorithm</td>
</tr>
<tr>
<td>coliny_beta</td>
<td>(Experimental) Coliny beta solver</td>
</tr>
<tr>
<td>nl2sol</td>
<td>Trust-region method for nonlinear least squares</td>
</tr>
<tr>
<td>nonlinear_cg</td>
<td>(Experimental) nonlinear conjugate gradient optimization</td>
</tr>
<tr>
<td>ncsu_direct</td>
<td>DIviding RECTangles method</td>
</tr>
<tr>
<td>genie_opt_darts</td>
<td>Voronoi-based high-dimensional global Lipschitzian optimization</td>
</tr>
<tr>
<td>genie_direct</td>
<td>Classical high-dimensional global Lipschitzian optimization</td>
</tr>
</tbody>
</table>
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>efficient_global</td>
<td>Global Surrogate Based Optimization, a.k.a. EGO</td>
</tr>
<tr>
<td>polynomial_chaos</td>
<td>Uncertainty quantification using polynomial chaos expansions</td>
</tr>
<tr>
<td>stoch_collocation</td>
<td>Uncertainty quantification with stochastic collocation</td>
</tr>
<tr>
<td>sampling</td>
<td>Randomly samples variables according to their distributions</td>
</tr>
<tr>
<td>importance_sampling</td>
<td>Importance sampling</td>
</tr>
<tr>
<td>gpais</td>
<td>Gaussian Process Adaptive Importance Sampling</td>
</tr>
<tr>
<td>adaptive_sampling</td>
<td>(Experimental) Build a GP surrogate and refine it adaptively</td>
</tr>
<tr>
<td>pof_darts</td>
<td>Probability-of-Failure (POF) darts is a novel method for estimating the probability of failure based on random sphere-packing.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rkd_darts</td>
<td>Recursive k-d (RKD) Darts: Recursive Hyperplane Sampling for Numerical Integration of High-Dimensional Functions.</td>
</tr>
<tr>
<td>efficient_subspace</td>
<td>(Experimental) efficient subspace method (ESM)</td>
</tr>
<tr>
<td>global_evidence</td>
<td>Evidence theory with evidence measures computed with global optimization methods</td>
</tr>
<tr>
<td>global_interval_est</td>
<td>Interval analysis using global optimization methods</td>
</tr>
<tr>
<td>bayes_calibration</td>
<td>Bayesian calibration</td>
</tr>
<tr>
<td>dace</td>
<td>Design and Analysis of Computer Experiments</td>
</tr>
<tr>
<td>fsu_cvt</td>
<td>Design of Computer Experiments - Centroidal Voronoi Tessellation</td>
</tr>
<tr>
<td>psuade_noat</td>
<td>Morris One-at-a-Time</td>
</tr>
<tr>
<td>local_evidence</td>
<td>Evidence theory with evidence measures computed with local optimization methods</td>
</tr>
</tbody>
</table>
Description

The `method` keyword signifies the start of a block in the Dakota input file. Said block contains the various keywords necessary to specify a method and to control its behavior.

**Method Block Requirements**

At least one `method` block must appear in the Dakota input file. Multiple `method` blocks may be needed to fully define advanced analysis approaches.

Each `method` block must specify one method and, optionally, any associated keywords that govern the behavior of the method.

**The Methods**

Each `method` block must select one method.

Starting with Dakota v6.0, the methods are grouped into two types: standard methods and multi-component methods.

The standard methods are stand-alone and self-contained in the sense that they only require a model to perform a study. They do not call other methods. While methods such as `polynomial_chaos` and `efficient_global` internally utilize multiple iterator and surrogate model components, these components are generally
hidden from user control due to restrictions on modularity; thus, these methods are stand-alone.

The multi-component group of methods provides a higher level "meta-algorithm" that points to other methods and models that support sub-iteration. For example, in a sequential hybrid method, the hybrid method specification must identify a list of subordinate methods, and the "meta-algorithm" executes these methods in sequence and transfers information between them. Surrogate-based minimizers provide another example in that they point both to other methods (e.g. what optimization method is used to solve the approximate subproblem) as well as to models (e.g. what type of surrogate model is employed). Multi-component methods generally provide some level of "plug and play" modularity, through their flexible support of a variety of method and model selections.

**Component-Based Iterator Commands**

Component-based iterator specifications include hybrid, multi-start, pareto set, surrogate-based local, surrogate-based global, and branch and bound methods. Whereas a standard iterator specification only needs an optional model pointer string (specified with model_pointer), component-based iterator specifications can include method pointer, method name, and model pointer specifications in order to define the components employed in the "meta-iteration." In particular, these specifications identify one or more methods (by pointer or by name) to specify the subordinate iterators that will be used in the top-level algorithm. Identifying a sub-iterator by name instead of by pointer is a lightweight option that relaxes the need for a separate method specification for the sub-iterator; however, a model pointer may be required in this case to provide the specification connectivity normally supported by the method pointer. Refer to these individual method descriptions for specific requirements for these advanced methods.

**Method Independent Controls**

In addition to the method, there are 10 optional keywords, which are referred to as method independent controls. These controls are valid for enough methods that it was reasonable to pull them out of the method dependent blocks and consolidate the specifications, however, they are NOT universally respected by all methods.

**Examples**

Several examples follow. The first example shows a minimal specification for an optimization method.

```
method
  dot_sqp
```

This example uses all of the defaults for this method.

A more sophisticated example would be

```
method,
  id_method = 'NLP1'
  dot_sqp
    max_iterations = 50
    convergence_tolerance = 1e-4
    output verbose
    model_pointer = 'M1'
```

This example demonstrates the use of identifiers and pointers as well as some method independent and method dependent controls for the sequential quadratic programming (SQP) algorithm from the DOT library. The max_iterations, convergence_tolerance, and output settings are method independent controls, in that they are defined for a variety of methods (see dot for usage of these controls).

The next example shows a specification for a least squares method.

```
method
  optpp_g_newton
    max_iterations = 10
    convergence_tolerance = 1.e-8
    search_method trust_region
    gradient_tolerance = 1.e-6
```
6.2. METHOD

Some of the same method independent controls are present along with several method dependent controls (search_method and gradient_tolerance) which are only meaningful for OPT++ methods (see package_optpp).

The next example shows a specification for a nondeterministic method with several method dependent controls (refer to sampling).

```plaintext
method
  sampling
  samples = 100
  seed = 12345
  sample_type lhs
  response_levels = 1000. 500.
```

The last example shows a specification for a parameter study method where, again, each of the controls are method dependent (refer to vector_parameter_study).

```plaintext
method
  vector_parameter_study
  step_vector = 1. 1. 1.
  num_steps = 10
```

6.2.1 id_method

- Keywords Area
- method
- id_method

Name the method block; helpful when there are multiple

Topics

This keyword is related to the topics:

- block_identifier
- method_independent_controls

Specification

Alias: none

Argument(s): STRING

Default: strategy use of last method parsed

Description

The method identifier string is supplied with id_method and is used to provide a unique identifier string for use with environment or meta-iterator specifications (refer to environment). It is appropriate to omit a method identifier string if only one method is included in the input file, since the single method to use is unambiguous in this case.
6.2.2 output

- Keywords Area
- method
- output

Control how much method information is written to the screen and output file

Topics

This keyword is related to the topics:

- dakota_output
- method_independent_controls

Specification

Alias: none  
Argument(s): none  
Default: normal

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output level (Group 1)</td>
<td>verbose</td>
<td>Level 4 of 5 - more than normal</td>
<td></td>
</tr>
<tr>
<td>output level (Group 1)</td>
<td>normal</td>
<td>Level 3 of 5 - default</td>
<td></td>
</tr>
<tr>
<td>output level (Group 1)</td>
<td>quiet</td>
<td>Level 2 of 5 - less than normal</td>
<td></td>
</tr>
<tr>
<td>output level (Group 1)</td>
<td>silent</td>
<td>Level 1 of 5 - minimum</td>
<td></td>
</tr>
</tbody>
</table>

Description

Choose from a total of five output levels during the course of a Dakota study. If there is no user specification for output verbosity, then the default setting is normal.

Specific mappings are as follows:

- silent (i.e., really quiet): silent iterators, silent model, silent interface, quiet approximation, quiet file operations
- quiet: quiet iterators, quiet model, quiet interface, quiet approximation, quiet file operations
- normal: normal iterators, normal model, normal interface, quiet approximation, quiet file operations
- verbose: verbose iterators, normal model, verbose interface, verbose approximation, verbose file operations
- debug (i.e., really verbose): debug iterators, normal model, debug interface, verbose approximation, verbose file operations
Note that iterators and interfaces utilize the full granularity in verbosity, whereas models, approximations, and file operations do not. With respect to iterator verbosity, different iterators implement this control in slightly different ways (as described below in the method independent controls descriptions for each iterator), however the meaning is consistent.

For models, interfaces, approximations, and file operations, quiet suppresses parameter and response set reporting and silent further suppresses function evaluation headers and scheduling output. Similarly, verbose adds file management, approximation evaluation, and global approximation coefficient details, and debug further adds diagnostics from nonblocking schedulers.

**debug**

- Keywords Area
- method
- output
- debug

Level 5 of 5 - maximum

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

This is described on output

**verbose**

- Keywords Area
- method
- output
- verbose

Level 4 of 5 - more than normal

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

This is described on output
normal

- Keywords Area
- method
- output
- normal

Level 3 of 5 - default

**Specification**

Alias: none
Argument(s): none

**Description**

This is described on output

quiet

- Keywords Area
- method
- output
- quiet

Level 2 of 5 - less than normal

**Specification**

Alias: none
Argument(s): none

**Description**

This is described on output

silent

- Keywords Area
- method
- output
- silent

Level 1 of 5 - minimum
6.2. METHOD

Specification

Alias: none
  Argument(s): none

Description
This is described on output

6.2.3 final_solutions

- Keywords Area
- method
- final_solutions

Number of designs returned as the best solutions

Topics
This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none
  Argument(s): INTEGER
  Default: 1

Description
The final_solutions controls the number of final solutions returned by the iterator as the best solutions.
  For most optimizers, this is one, but some optimizers can produce multiple solutions (e.g. genetic algorithms).
  When using a hybrid strategy, the number of final solutions dictates how many solutions are passed from one method to another.

Examples
In the case of sampling methods, if one specifies 100 samples (for example) but also specifies final_solutions = 5, the five best solutions (in order of lowest response function value) are returned.

6.2.4 hybrid

- Keywords Area
- method
- hybrid

Strategy in which a set of methods synergistically seek an optimal design
## Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required (Choose One)</strong></td>
<td>Group 1</td>
<td>sequential</td>
<td>Methods are run one at a time, in sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>embedded</td>
<td>A subordinate local method provides periodic refinements to a top-level global method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>collaborative</td>
<td>Multiple methods run concurrently and share information</td>
</tr>
<tr>
<td>Optional</td>
<td>iterator_servers</td>
<td>Specify the number of iterator servers when Dakota is run in parallel</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>iterator_scheduling</td>
<td>Specify the scheduling of concurrent iterators when Dakota is run in parallel</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>processors_per_iterator</td>
<td>Specify the number of processors per iterator server when Dakota is run in parallel</td>
<td></td>
</tr>
</tbody>
</table>

## Description

In a hybrid minimization method (hybrid), a set of methods synergistically seek an optimal design. The relationships among the methods are categorized as:

- collaborative
- embedded
- sequential

The goal in each case is to exploit the strengths of different optimization and nonlinear least squares algorithms at different stages of the minimization process. Global + local hybrids (e.g., genetic algorithms combined with nonlinear programming) are a common example in which the desire for identification of a global optimum is balanced with the need for efficient navigation to a local optimum.
6.2. METHOD

sequential

- Keywords Area
- method
- hybrid
- sequential

Methods are run one at a time, in sequence

Specification

Alias: uncoupled
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>method_name_list</td>
<td>List of Dakota methods to sequentially or collaboratively run</td>
</tr>
<tr>
<td>Choose One</td>
<td></td>
<td>method_pointer_list</td>
<td>Pointers to methods to execute sequentially or collaboratively</td>
</tr>
</tbody>
</table>

Description

In the sequential approach, methods are run one at a time, in sequence. The best solutions from one method are used to initialize the next method.

The sequence of methods (i.e. iterators) to run are specified using either a method_pointer_list or a method_name_list (with optional model_pointer_list). Any number of iterators may be specified.

Method switching is managed through the separate convergence controls of each method. The number of solutions transferred between methods is specified by the particular method through its final_solutions method control.

For example, if one sets up a two-level study with a first method that generates multiple solutions such as a genetic algorithm, followed by a second method that is initialized only at a single point such as a gradient-based algorithm, it is possible to take the multiple solutions generated by the first method and create several instances of the second method, each one with a different initial starting point.

The logic governing the transfer of multiple solutions between methods is as follows:

- if one solution is returned from method A, then one solution is transferred to method B.

- If multiple solutions are returned from method A, and method B can accept multiple solutions as input (for example, as a genetic algorithm population), then one instance of method B is initialized with multiple solutions.

- If multiple solutions are returned from method A but method B only can accept one initial starting point, then method B is run several times, each one with a separate starting point from the results of method A.
**method_name_list**

- Keywords Area
- method
- hybrid
- sequential
- method_name_list

List of Dakota methods to sequentially or collaboratively run

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>model_pointer_list</td>
<td>Associate models with method names</td>
</tr>
</tbody>
</table>

**Description**

*method_name_list* specifies a list of Dakota methods (e.g. *soga, conmin_frcg*) that will be run by a hybrid sequential or hybrid collaborative method. The methods are executed with default options. The optional *model_pointer_list* may be used to associate a model with each method.

**model_pointer_list**

- Keywords Area
- method
- hybrid
- sequential
- method_name_list
- model_pointer_list

Associate models with method names

**Topics**

This keyword is related to the topics:

- block_pointer

**Specification**

**Alias:** none

**Argument(s):** STRING
6.2. **METHOD**

**Description**

Using the optional keyword `model_pointer_list`, models can be assigned to methods specified in the `method_name_list`. Models are referred to by name (i.e. by their `id_model` labels). The length of the `model_pointer_list` must be either 1 or match the length of the `method_name_list`. If the former, the same model will be used for all methods, and if the latter, methods and models will be paired in the order that they appear in the two lists.

**method_pointer_list**

- **Keywords Area**
- method
- hybrid
- sequential
- `method_pointer_list`

Pointers to methods to execute sequentially or collaboratively

**Topics**

This keyword is related to the topics:

- `block_pointer`

**Specification**

*Alias*: none

*Argument(s)*: `STRINGLIST`

**Description**

`method_pointer_list` specifies by name the methods that are to be executed by a hybrid sequential or hybrid collaborative method. Its argument is a list of strings that refer to method blocks by name (i.e. to their `id_method` labels).

**embedded**

- **Keywords Area**
- method
- hybrid
- `embedded`

A subordinate local method provides periodic refinements to a top-level global method

**Specification**

*Alias*: `coupled`

*Argument(s)*: none
### Description

In the embedded approach, a tightly-coupled hybrid is employed in which a subordinate local method provides periodic refinements to a top-level global method.

Global and local method strings supplied with the `global_method_pointer` and `local_method_pointer` specifications identify the two methods to be used. Alternatively, Dakota method names (e.g. 'soga') can be supplied using the `global_method_name` and `local_method_name` keywords, which each have optional model pointer specifications. The `local_search_probability` setting is an optional specification for supplying the probability (between 0.0 and 1.0) of employing local search to improve estimates within the global search.

**global_method_name**

- Keywords Area
- method
- hybrid
- embedded
- `global_method_name`

Specify the global method by Dakota name

### Specification

**Alias:** none

**Argument(s):** STRING

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> (Choose One)</td>
<td>global method name</td>
<td>global_method_pointer</td>
<td>Specify the global method by Dakota name</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>Group 2</td>
<td>local_method name</td>
<td>local_method_pointer</td>
</tr>
<tr>
<td>Optional</td>
<td>local_search_probability</td>
<td>local_search_probability</td>
<td>Probability of executing local searches</td>
</tr>
</tbody>
</table>

---

**Notes:**

- The embedded approach is designed to enhance the global optimization process by incorporating local search refinement.
- Dakota method names can be specified for both global and local methods, providing flexibility in choosing the optimization algorithms.
- The local search probability allows users to control the frequency of local search evaluations, which can be crucial for improving the accuracy and efficiency of the overall optimization process.

---

**Example Usage:**

In a Dakota input file, you might specify the global method name as follows:

```
method hybrid
  embedded
  global_method_name: 'soga'
```

This configuration indicates that the Dakota optimization process will use the 'soga' method as the global method, embedded within a hybrid approach that includes embedded local searches with a specified probability.
Description

global_method_name is used to specify the global method in a hybrid embedded optimization by Dakota name (e.g. 'soga'). The name of the method is provided as a string. The method is executed with default options.

**global_model_pointer**

- Keywords Area
- method
- hybrid
- embedded
- global_method_name
- global_model_pointer

Pointer to model used by global method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Description

global_model_pointer can be used to specify a model for use with the Dakota method named by the global_method_name specification. The argument is a string that refers to the id_model label of the desired model.

**global_method_pointer**

- Keywords Area
- method
- hybrid
- embedded
- global_method_pointer

Pointer to global method
CHAPTER 6. KEYWORDS AREA

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Description

The `global_method_pointer` identifies the method block to use as the global method in a hybrid embedded optimization using its `id_method` label.

local_method_name

- Keywords Area
- method
- hybrid
- embedded
- local_method_name

Specify the local method by Dakota name

Specification

Alias: none

Argument(s): STRING

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>local_model_pointer</td>
<td>Pointer to model used by local method</td>
</tr>
</tbody>
</table>

Description

local_method_name is used to specify the local method in a hybrid embedded optimization by Dakota name (e.g. `conmin_mfd`). The name of the method is provided as a string. The method is executed with default options.

local_model_pointer

- Keywords Area
- method
- hybrid
6.2. METHOD

- embedded
- local_method_name
- local_model_pointer

Pointer to model used by local method

Topics
This keyword is related to the topics:
- block_pointer

Specification
Alias: none
Argument(s): STRING

Description
local_model_pointer can be used to specify a model for use with the Dakota method named by the local-method_name specification. The argument is a string that refers to the id_model label of the desired model.

local_method_pointer
- Keywords Area
- method
- hybrid
- embedded
- local_method_pointer

Pointer to local method

Topics
This keyword is related to the topics:
- block_pointer

Specification
Alias: none
Argument(s): STRING

Description
The local_method_pointer identifies the method block to use as the local method in a hybrid embedded optimization using its id_method
local_search_probability

- Keywords Area
- method
- hybrid
- embedded
- local_search_probability

Probability of executing local searches

Specification

Alias: none
Argument(s): REAL

Description

The local_search_probability setting is an optional specification for supplying the probability (between 0.0 and 1.0) of employing local search to improve estimates within the global search. Its default value is 0.1.

collaborative

- Keywords Area
- method
- hybrid
- collaborative

Multiple methods run concurrently and share information

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword method_name_list</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>List of Dakota methods to sequentially or collaboratively run</td>
</tr>
</tbody>
</table>
6.2. METHOD

Description

In the collaborative approach, multiple methods work together and share solutions while executing concurrently. A list of method strings specifies the pool of iterators to be used. Any number of iterators may be specified. The method collaboration logic follows that of either the Agent-Based Optimization or HOPSPACK codes and is currently under development and not available at this time.

method_name_list

- Keywords Area
- method
- hybrid
- collaborative
- method_name_list

List of Dakota methods to sequentially or collaboratively run

Specification

Alias: none

Argument(s): STRINGLIST

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>model_pointer_list</td>
<td>Associate models with method names</td>
</tr>
</tbody>
</table>

Description

method_name_list specifies a list of Dakota methods (e.g. soga, conmin_frcg) that will be run by a hybrid sequential or hybrid collaborative method. The methods are executed with default options. The optional model_pointer_list may be used to associate a model with each method.

model_pointer_list

- Keywords Area
- method
- hybrid
- collaborative
- method_name_list
- model_pointer_list

Associate models with method names
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING

Description
Using the optional keyword model_pointer_list, models can be assigned to methods specified in the method_name_list. Models are referred to by name (i.e. by their id_model labels). The length of the model_pointer_list must be either 1 or match the length of the method_name_list. If the former, the same model will be used for all methods, and if the latter, methods and models will be paired in the order that they appear in the two lists.

method_pointer_list

- Keywords Area
- method
- hybrid
- collaborative
- method_pointer_list

Pointers to methods to execute sequentially or collaboratively

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRINGLIST

Description
method_pointer_list specifies by name the methods that are to be executed by a hybrid sequential or hybrid collaborative method. Its argument is a list of strings that refer to method blocks by name (i.e. to their id_method labels).
6.2. METHOD

iterator_servers
- Keywords Area
- method
- hybrid
- iterator_servers

Specify the number of iterator servers when Dakota is run in parallel

Topics
This keyword is related to the topics:
- concurrency_and_parallelism

Specification
Alias: none
Argument(s): INTEGER

Description
An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional iterator_servers specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

iterator_scheduling
- Keywords Area
- method
- hybrid
- iterator_scheduling

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics
This keyword is related to the topics:
- concurrency_and_parallelism

Specification
Alias: none
Argument(s): none
### Keywords Area

<table>
<thead>
<tr>
<th>Required/Optional Group</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td>master</td>
<td>Specify a dedicated master partition for parallel iterator scheduling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>peer</td>
<td>Specify a peer partition for parallel iterator scheduling</td>
</tr>
</tbody>
</table>

### Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional *iterator_scheduling* specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, *hybrid*, *multi_start*, and *pareto_set* component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

#### master

- **Keywords Area**
- **method**
- **hybrid**
- **iterator_scheduling**
- **master**

Specify a dedicated master partition for parallel iterator scheduling

### Topics

This keyword is related to the topics:

- `concurrency_and_parallelism`

### Specification

**Alias**: none

**Argument(s)**: none

### Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.
6.2. METHOD

peer

- Keywords Area
- method
- hybrid
- iterator_scheduling
- peer

Specify a peer partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of evaluation_scheduling, it is not possible to specify static or dynamic.

processors_per_iterator

- Keywords Area
- method
- hybrid
- processors_per_iterator

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none

Argument(s): INTEGER
Description
An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional processors_per_iterator specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual[4] for additional information.

6.2.5 multi_start

- Keywords Area
- method
- multi_start

Multi-Start Optimization Method

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required (Choose One)</strong></td>
<td></td>
<td>method_name</td>
<td>Specify sub-method by name</td>
</tr>
<tr>
<td>Optional</td>
<td>method_pointer</td>
<td></td>
<td>Pointer to sub-method to run from each starting point</td>
</tr>
<tr>
<td>Optional</td>
<td>random_starts</td>
<td></td>
<td>Number of random starting points</td>
</tr>
<tr>
<td>Optional</td>
<td>starting_points</td>
<td></td>
<td>List of user-specified starting points</td>
</tr>
<tr>
<td>Optional</td>
<td>iterator_servers</td>
<td></td>
<td>Specify the number of iterator servers when Dakota is run in parallel</td>
</tr>
</tbody>
</table>
6.2. METHOD

| Optional | iterator_scheduling | Specify the scheduling of concurrent iterators when Dakota is run in parallel |
| Optional | processors_per_iterator | Specify the number of processors per iterator server when Dakota is run in parallel |

Description

In the multi-start iteration method (`multi_start`), a series of iterator runs are performed for different values of parameters in the model. A common use is for multi-start optimization (i.e., different local optimization runs from different starting points for the design variables), but the concept and the code are more general. Multi-start iteration is implemented within the MetaIterator branch of the Iterator hierarchy within the ConcurrentMetaIterator class. Additional information on the multi-start algorithm is available in the Users Manual[4].

The `multi_start` meta-iterator must specify a sub-iterator using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This iterator is responsible for completing a series of iterative analyses from a set of different starting points. These starting points can be specified as follows: (1) using `random_starts`, for which the specified number of starting points are selected randomly within the variable bounds, (2) using `starting_points`, in which the starting values are provided in a list, or (3) using both `random_starts` and `starting_points`, for which the combined set of points will be used. In aggregate, at least one starting point must be specified. The most common example of a multi-start algorithm is multi-start optimization, in which a series of optimizations are performed from different starting values for the design variables. This can be an effective approach for problems with multiple minima.

**method_name**

- **Keywords Area**
- **method**
- **multi_start**
- **method_name**

Specify sub-method by name

**Specification**

**Alias:** none

**Argument(s):** STRING

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
</table>
### Description

The `method_name` keyword is used to specify a sub-method by Dakota method name (e.g. ’npsol_sqp’) rather than block pointer. The method will be executed using its default settings. The optional `model_pointer` specification can be used to associate a model block with the method.

#### model_pointer

- **Keywords Area**
- **method**
- **multi_start**
- **method_name**
- **model_pointer**

Identifier for model block to be used by a method

#### Topics

This keyword is related to the topics:

- **block_pointer**

#### Specification

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

#### Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative. See `block_pointer` for details about pointers.
6.2. METHOD

Examples

environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
1.0 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

method_pointer

- Keywords Area
- method
- multi_start
- method_pointer

Pointer to sub-method to run from each starting point
Topics
This keyword is related to the topics:

- **block_pointer**

Specification
Alias: none
Argument(s): STRING

Description
The `method_pointer` keyword is used to specify a pointer to the sub-method block that will be run from each starting point.

**random_starts**

- **Keywords Area**
- **method**
- **multi_start**
- **random_starts**

Number of random starting points

Specification
Alias: none
Argument(s): INTEGER

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
</tbody>
</table>

Description
The `multi_start` meta-iterator must specify a sub-iterator using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This iterator is responsible for completing a series of iterative analyses from a set of different starting points. These starting points can be specified as follows: (1) using `random_starts`, for which the specified number of starting points are selected randomly within the variable bounds, (2) using `starting_points`, in which the starting values are provided in a list, or (3) using both `random_starts` and `starting_points`, for which the combined set of points will be used.
6.2. METHOD

seed

- Keywords Area
- method
- multi_start
- random_starts
- seed

Seed of the random number generator

Specification

Alias: none
Argument(s): INTEGER
Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

starting_points

- Keywords Area
- method
- multi_start
- starting_points

List of user-specified starting points
**CHAPTER 6. KEYWORDS AREA**

**Specification**

Alias: none

Argument(s): REALLIST

**Description**

The `multi_start` meta-iterator must specify a sub-iterator using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This iterator is responsible for completing a series of iterative analyses from a set of different starting points. These starting points can be specified as follows: (1) using `random_starts`, for which the specified number of starting points are selected randomly within the variable bounds, (2) using `starting_points`, in which the starting values are provided in a list, or (3) using both `random_starts` and `starting_points`, for which the combined set of points will be used.

**iterator_servers**

- Keywords Area
- method
- multi_start
- iterator_servers

Specify the number of iterator servers when Dakota is run in parallel

**Topics**

This keyword is related to the topics:

- `concurrency_and_parallelism`

**Specification**

Alias: none

Argument(s): INTEGER

**Description**

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_servers` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

**iterator_scheduling**

- Keywords Area
- method
- multi_start
• iterator_scheduling

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics
This keyword is related to the topics:
• concurrency_and_parallelism

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>master</td>
<td>Specify a dedicated master partition for parallel iterator scheduling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>peer</td>
<td>Specify a peer partition for parallel iterator scheduling</td>
</tr>
</tbody>
</table>

Description
An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional iterator_scheduling specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

master
• Keywords Area
• method
• multi_start
• iterator_scheduling
• master
Specify a dedicated master partition for parallel iterator scheduling

Topics
This keyword is related to the topics:
• concurrency_and_parallelism
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
Argument(s): none

Description
This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

peer

- Keywords Area
- method
- multi_start
- iterator_scheduling
- peer

Specify a peer partition for parallel iterator scheduling

Topics
This keyword is related to the topics:
- concurrency_and_parallelism

Specification
Alias: none
Argument(s): none

Description
This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of evaluation_scheduling, it is not possible to specify static or dynamic.

processors_per_iterator

- Keywords Area
- method
- multi_start
- processors_per_iterator

Specify the number of processors per iterator server when Dakota is run in parallel
6.2. METHOD

Topics
This keyword is related to the topics:

- concurrency_and_parallelism

Specification
Alias: none
Argument(s): INTEGER

Description
An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional processors_per_iterator specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual[4] for additional information.

6.2.6 pareto_set

- Keywords Area
- method
- pareto_set

Pareto set optimization

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method_name</td>
<td>Group 1</td>
<td>method_pointer</td>
<td>Pointer to optimization or least-squares sub-method</td>
</tr>
</tbody>
</table>
**Keywords Area**

<table>
<thead>
<tr>
<th>Optional</th>
<th>random_weight_sets</th>
<th>Number of random weighting sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>weight_sets</td>
<td>List of user-specified weighting sets</td>
</tr>
<tr>
<td>Optional</td>
<td>iterator_servers</td>
<td>Specify the number of iterator servers when Dakota is run in parallel</td>
</tr>
<tr>
<td>Optional</td>
<td>iterator_scheduling</td>
<td>Specify the scheduling of concurrent iterators when Dakota is run in parallel</td>
</tr>
<tr>
<td>Optional</td>
<td>processors_per_iterator</td>
<td>Specify the number of processors per iterator server when Dakota is run in parallel</td>
</tr>
</tbody>
</table>

**Description**

In the pareto set minimization method (pareto_set), a series of optimization or least squares calibration runs are performed for different weightings applied to multiple objective functions. This set of optimal solutions defines a "Pareto set," which is useful for investigating design trade-offs between competing objectives. The code is similar enough to the multi_start technique that both algorithms are implemented in the same Concurrent-MetaIterator class.

The pareto_set specification must identify an optimization or least squares calibration method using either a method_pointer or a method_name plus optional model_pointer. This minimizer is responsible for computing a set of optimal solutions from a set of response weightings (multi-objective weights or least squares term weights). These weightings can be specified as follows: (1) using random_weight_sets, in which case weightings are selected randomly within [0,1] bounds, (2) using weight_sets, in which the weighting sets are specified in a list, or (3) using both random_weight_sets and weight_sets, for which the combined set of weights will be used. In aggregate, at least one set of weights must be specified. The set of optimal solutions is called the "pareto set," which can provide valuable design trade-off information when there are competing objectives.

**method_name**

- Keywords Area
- method
- pareto_set
- method_name

Specify sub-method by name
6.2. METHOD

Specification

Alias: opt_method_name

Argument(s): STRING

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional   | Group          |               | Description   |
|           | model_pointer  | Identifier for | model block to |
|           |                | be used by a   | be used by a  |
|           |                | method         | method        |

Description

The method_name keyword is used to specify a sub-method by Dakota method name (e.g. `npsol_sqp`) rather than block pointer. The method will be executed using its default settings. The optional model_pointer specification can be used to associate a model block with the method.

model_pointer

- Keywords Area
- method
- pareto_set
- method_name
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: opt_model_pointer

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.
Examples

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling
  samples = 10
  seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
  0.1 0.2 0.6
  0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

method_pointer
  • Keywords Area
  • method
  • pareto_set
  • method_pointer

  Pointer to optimization or least-squares sub-method
```
6.2. METHOD

Topics
This keyword is related to the topics:

- block_pointer

Specification

Alias: opt_method_pointer
Argument(s): STRING

Description
The method_pointer keyword is used to specify a pointer to an optimization or least-squares sub-method that is responsible for computing a set of optimal solutions for a set of response weightings.

random_weight_sets

- Keywords Area
- method
- pareto_set
- random_weight_sets

Number of random weighting sets

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>Seed of the random number generator</td>
</tr>
</tbody>
</table>

Description
The pareto_set specification must identify an optimization or least squares calibration method using either a method_pointer or a method_name plus optional model_pointer. This minimizer is responsible for computing a set of optimal solutions from a set of response weightings (multi-objective weights or least squares term weights). These weightings can be specified as follows: (1) using random_weight_sets, in which case weightings are selected randomly within [0,1] bounds, (2) using weight_sets, in which the weighting sets are specified in a list, or (3) using both random_weight_sets and weight_sets, for which the combined set of weights will be used. In aggregate, at least one set of weights must be specified. The set of optimal solutions is called the "pareto set," which can provide valuable design trade-off information when there are competing objectives.
CHAPTER 6. KEYWORDS AREA

seed

- Keywords Area
- method
- pareto_set
- random_weight_sets
- seed

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

weight_sets

- Keywords Area
- method
- pareto_set
- weight_sets

List of user-specified weighting sets
6.2. METHOD

Specification

Alias: multi_objective_weight_sets
Argument(s): REALLIST

Description

The pareto_set specification must identify an optimization or least squares calibration method using either a method_pointer or a method_name plus optional model_pointer. This minimizer is responsible for computing a set of optimal solutions from a set of response weightings (multi-objective weights or least squares term weights). These weightings can be specified as follows: (1) using random_weight_sets, in which case weightings are selected randomly within [0,1] bounds, (2) using weight_sets, in which the weighting sets are specified in a list, or (3) using both random_weight_sets and weight_sets, for which the combined set of weights will be used. In aggregate, at least one set of weights must be specified. The set of optimal solutions is called the "pareto set," which can provide valuable design trade-off information when there are competing objectives.

iterator_servers

- Keywords Area
- method
- pareto_set
- iterator_servers

Specify the number of iterator servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none
Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional iterator_servers specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.
iterator_scheduling

- Keywords Area
- method
- pareto_set
- iterator_scheduling

Specify the scheduling of concurrent iterators when Dakota is run in parallel.

Topics

This keyword is related to the topics:
- concurrency_and_parallelism

Specification

Alias: none  
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td></td>
<td></td>
<td>Specify a dedicated master partition for parallel iterator scheduling</td>
</tr>
<tr>
<td>peer</td>
<td></td>
<td></td>
<td>Specify a peer partition for parallel iterator scheduling</td>
</tr>
</tbody>
</table>

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional iterator_scheduling specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterations. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

master

- Keywords Area
- method
- pareto_set
- iterator_scheduling
- master

Specify a dedicated master partition for parallel iterator scheduling
6.2. METHOD

Topics
This keyword is related to the topics:

- concurrency_and_parallelism

Specification
Alias: none
Argument(s): none

Description
This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

peer

- Keywords Area
- method
- pareto_set
- iterator_scheduling
- peer

Specify a peer partition for parallel iterator scheduling

Topics
This keyword is related to the topics:

- concurrency_and_parallelism

Specification
Alias: none
Argument(s): none

Description
This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of evaluation_scheduling, it is not possible to specify static or dynamic.
processors_per_iterlator

- Keywords Area
- method
- pareto_set
- processors_per_iterator

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:
- concurrency_and_parallelism

Specification

Alias: none
Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterlator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterator runs. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual[4] for additional information.

6.2.7 branch_and_bound

- Keywords Area
- method
- branch_and_bound

(Experimental Capability) Solves a mixed integer nonlinear optimization problem

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
</table>
6.2. METHOD

### Required (Choose One)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>method_pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group 1</th>
<th>method_name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specify sub-method by name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group 1</th>
<th>scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
</tbody>
</table>

### Description

The branch-and-bound optimization methods solve mixed integer nonlinear optimization problems. It does so by partitioning the parameter space according to some criteria along the integer or discrete variables. It then relaxes (i.e., treats all variables as continuous) the sub-problems created by the partitions and solves each sub-problem with a continuous nonlinear optimization method. Results of the sub-problems are combined in such a way that yields the solution to the original optimization problem.

### Default Behavior

Branch-and-bound expects all discrete variables to be relaxable. If your problem has categorical or otherwise non-relaxable discrete variables, then this is not the optimization method you are looking for.

### Expected Output

The optimal solution and associated parameters will be printed to the screen output.

### Usage Tips

The user must choose a nonlinear optimization method to solve the sub-problems. We recommend choosing a method that would be chosen to solve a continuous problem that has similar form to the mixed integer problem.

### Examples

```plaintext
environment
method_pointer = 'BandB'

method
id_method = 'BandB'
branch_and_bound
output verbose
method_pointer = 'SubNLP'

method
id_method = 'SubNLP'
coliny_ea
    seed = 12345
    max_iterations = 100
    max_function_evaluations = 100

variables,
continuous_design = 3
initial_point -1.0 1.5 2.0
upper_bounds 10.0 10.0 10.0
lower_bounds -10.0 -10.0 -10.0
```
Keywords Area

• Keywords Area

• method

• branch_and_bound

• method_pointer

Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem

Topics

This keyword is related to the topics:

• block_pointer

Specification

Alias: none

Argument(s): STRING

Description

The method_pointer keyword is used to specify a pointer to an optimization or least-squares sub-method to apply in the context of what could be described as hierarchical methods. In surrogate-based methods, the sub-method is applied to the surrogate model. In the branch-and-bound method, the sub-method is applied to the relaxed sub-problems.

Any model_pointer identified in the sub-method specification is ignored. Instead, the parent method is responsible for selecting the appropriate model to use as specified by its model_pointer. In surrogate-based methods, it is a surrogate model defined using its model_pointer. In branch-and-bound methods, it is the relaxed model that is constructed internally from the original model.
method_name
  • Keywords Area
  • method
  • branch_and_bound
  • method_name

Specify sub-method by name

Specification

**Alias:** none

**Argument(s):** STRING

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

The `method_name` keyword is used to specify a sub-method by Dakota method name (e.g. ‘npsol_sqp’) rather than block pointer. The method will be executed using its default settings. The optional `model_pointer` specification can be used to associate a model block with the method.

model_pointer
  • Keywords Area
  • method
  • branch_and_bound
  • method_name
  • model_pointer

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:
  • block_pointer

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYS

Description

The **model_pointer** is used to specify which **model** block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a **model** block in the Dakota input file that has a corresponding **id_model** with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a **model-pointer** for each method is imperative.

See **block_pointer** for details about pointers.

**Examples**

```
environment
   tabular_graphics_data
   method_pointer = 'UQ'

method
   id_method = 'UQ'
   model_pointer = 'SURR'
   sampling,
      samples = 10
      seed = 98765 rng rnum2
      response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
      0.1 0.2 0.6
   sample_type lhs
   distribution cumulative

model
   id_model = 'SURR'
   surrogate global,
      dace_method_pointer = 'DACE'
      polynomial quadratic

method
   id_method = 'DACE'
   model_pointer = 'DACE_M'
   sampling sample_type lhs
      samples = 121 seed = 5034 rng rnum2

model
   id_model = 'DACE_M'
   single
   interface_pointer = 'I1'

variables
   uniform_uncertain = 2
      lower_bounds = 0. 0.
      upper_bounds = 1. 1.
   descriptors = 'x1' 'x2'

interface
   id_interface = 'I1'
   system asynch evaluation_concurrency = 5
      analysis_driver = 'text_book'

responses
```
Scaling

- **Keywords Area**
- **method**
- **branch_and_bound**
- **scaling**

Turn on scaling for variables, responses, and constraints

**Topics**

This keyword is related to the topics:

- **method**, **independent_controls**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no scaling

**Description**

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the **method**, **variables**, and **responses** blocks. When the `scaling` keyword is omitted, all `*_scale_types` and `*_scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `*scales` keyword, which gives characteristic values
- a `*scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. none (default) - no scaling, value of `*scales` keyword is ignored

response_functions = 3
no_gradients
no_hessians
2. **value** - multiplicative scaling

3. **auto** - automatic scaling

   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according
   to the following scheme:

   • two-sided bounds scaled into the interval $[0,1]$;
   • one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and
     changing the sense of inequalities where necessary;
   • no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound con-
   straints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not
   affinely scaled into $[0,1]$.

4. **log** - logarithmic scaling

   First, any characteristic values from the optional $\ast_{\text{scales}}$ specification are applied. Then logarithm base
   10 scaling is applied.

   Logarithmic scaling is not available for linear constraints.

   When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The $\ast_{\text{scales}}$ keywords are used to specify the characteristic values. These must be non-zero real numbers.

The numbers are used according to the corresponding $\ast_{\text{scale_type}}$, as described above.

Depending on the scale type, the characteristic values may be required or optional.

   • none, auto, log - optional

   • value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10*\text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```plaintext
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100

responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```
6.2. METHOD

6.2.8 surrogate_based_local

- Keywords Area
- method
- surrogate_based_local

Local Surrogate Based Optimization

Topics

This keyword is related to the topics:

- surrogate_based_optimization_methods

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>method_pointer</td>
<td>method_name</td>
<td>Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem</td>
</tr>
<tr>
<td>Required</td>
<td>model_pointer</td>
<td></td>
<td>Identifier for model block to be used by a method</td>
</tr>
<tr>
<td>Optional</td>
<td>soft_convergence_limit</td>
<td></td>
<td>Limit number of iterations w/ little improvement</td>
</tr>
<tr>
<td>Optional</td>
<td>truth_surrogate_bypass</td>
<td></td>
<td>Bypass lower level surrogates when performing truth verifications on a top level surrogate</td>
</tr>
</tbody>
</table>
Optional

trust_region

Use trust region search method

Optional

approx_-subproblem

Identify functions to be included in surrogate merit function

Optional

merit_function

Select type of penalty or merit function

Optional

acceptance_logic

Set criteria for trusted surrogate

Optional

constraint_relax

Enable constraint relaxation

Optional

max_iterations

Stopping criterion based on number of iterations

Optional

convergence_-tolerance

Stopping criterion based on convergence of the objective function or statistics

Optional

constraint_-tolerance

The maximum allowable value of constraint violation still considered to be feasible

**Description**

In surrogate-based optimization (SBO) and surrogate-based nonlinear least squares (SBNLS), minimization occurs using a set of one or more approximations, defined from a surrogate model, that are built and periodically updated using data from a "truth" model. The surrogate model can be a global data fit (e.g., regression or interpolation of data generated from a design of computer experiments), a multipoint approximation, a local Taylor Series expansion, or a model hierarchy approximation (e.g., a low-fidelity simulation model), whereas the truth model involves a high-fidelity simulation model. The goals of surrogate-based methods are to reduce the total number of truth model simulations and, in the case of global data fit surrogates, to smooth noisy data with an easily navigated analytic function.

In the surrogate-based local method, a trust region approach is used to manage the minimization process to maintain acceptable accuracy between the surrogate model and the truth model (by limiting the range over which the surrogate model is trusted). The process involves a sequence of minimizations performed on the surrogate model and bounded by the trust region. At the end of each approximate minimization, the candidate optimum point is validated using the truth model. If sufficient decrease has been obtained in the truth model, the trust region is re-centered around the candidate optimum point and the trust region will either shrink, expand, or remain the same size depending on the accuracy with which the surrogate model predicted the truth model decrease. If sufficient decrease has not been attained, the trust region center is not updated and the entire trust region shrinks by a user-specified factor. The cycle then repeats with the construction of a new surrogate model, a minimization, and another test for sufficient decrease in the truth model. This cycle continues until convergence is attained.
6.2. METHOD

Theory
For surrogate_based_local problems with nonlinear constraints, a number of algorithm formulations exist as described in[23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual[4].

See Also
These keywords may also be of interest:

- efficient_global
- surrogate_based_global

method_pointer

- Keywords Area
- method
- surrogate_based_local
- method_pointer

   Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem

Topics
This keyword is related to the topics:

- block_pointer

Specification

Alias: approx_method_pointer
Argument(s): STRING

Description
The method_pointer keyword is used to specify a pointer to an optimization or least-squares sub-method to apply in the context of what could be described as hierarchical methods. In surrogate-based methods, the sub-method is applied to the surrogate model. In the branch-and-bound method, the sub-method is applied to the relaxed sub-problems.

Any model_pointer identified in the sub-method specification is ignored. Instead, the parent method is responsible for selecting the appropriate model to use as specified by its model_pointer. In surrogate-based methods, it is a surrogate model defined using its model_pointer. In branch-and-bound methods, it is the relaxed model that is constructed internally from the original model.
method_name

- Keywords Area
- method
- surrogate_based_local
- method_name

Specify sub-method by name

Specification

Alias: approx_method_name
Argument(s): STRING

Description

The method_name keyword is used to specify a sub-method by Dakota method name (e.g. ’npsol_sqp’) rather than block pointer. The method will be executed using its default settings. The optional model_pointer specification can be used to associate a model block with the method.

model_pointer

- Keywords Area
- method
- surrogate_based_local
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: approx_model_pointer
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

```plaintext
environment
tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SRR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
  0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SRR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
```
response_functions = 3
no_gradients
no_hessians

**soft_convergence_limit**

- **Keywords Area**
- **method**
- **surrogate_based_local**
- **soft_convergence_limit**

Limit number of iterations w/ little improvement

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 5

**Description**

**soft_convergence_limit** (a soft convergence control for the surrogate_based_local iterations which limits the number of consecutive iterations with improvement less than the convergence tolerance)

**truth_surrogate_bypass**

- **Keywords Area**
- **method**
- **surrogate_based_local**
- **truth_surrogate_bypass**

Bypass lower level surrogates when performing truth verifications on a top level surrogate

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no bypass

**Description**

**truth_surrogate_bypass** (a flag for bypassing all lower level surrogates when performing truth verifications on a top level surrogate).
6.2. METHOD

trust_region

- Keywords Area
- method
- surrogate_based_local
- trust_region

Use trust region search method

Specification

Alias: none
Argument(s): none

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional   | Group         |               | Description   |
| Optional   | initial_size  | initial_size  | Trust region initial size (relative to bounds) |
| Optional   | minimum_size  | minimum_size  | Trust region minimum size |
| Optional   | contract_threshold | contract_threshold | Shrink trust region if trust region ratio is below this value |
| Optional   | expand_threshold | expand_threshold | Expand trust region if trust region ratio is above this value |
| Optional   | contraction_factor | contraction_factor | Amount by which step length is rescaled |
| Optional   | expansion_factor | expansion_factor | Trust region expansion factor |

Description

The trust_region optional group specification can be used to specify the initial size of the trust region (using initial_size) relative to the total variable bounds, the minimum size of the trust region (using minimum_size), the contraction factor for the trust region size (using contraction_factor) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using expansion_factor) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using contract_threshold) and the trust region size expansion threshold (using expand_threshold). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command contract_threshold sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next surrogate_based_local iteration. The command expand_threshold determines the trust region value above which the trust region will expand for the next surrogate_based_local iteration.
initial_size

- Keywords Area
- method
- surrogate_based_local
- trust_region
- initial_size

Trust region initial size (relative to bounds)

Specification

Alias: none
- Argument(s): REAL
- Default: 0.4

Description

The trust_region optional group specification can be used to specify the initial size of the trust region (using initial_size) relative to the total variable bounds, the minimum size of the trust region (using minimum_size), the contraction factor for the trust region size (using contraction_factor) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using expansion_factor) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using contract_threshold) and the trust region size expansion threshold (using expand_threshold). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command contract_threshold sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command expand_threshold determines the trust region value above which the trust region will expand for the next SBL iteration.

minimum_size

- Keywords Area
- method
- surrogate_based_local
- trust_region
- minimum_size

Trust region minimum size

Specification

Alias: none
- Argument(s): REAL
- Default: 1.e-6
Description

The trust_region optional group specification can be used to specify the initial size of the trust region (using initial_size) relative to the total variable bounds, the minimum size of the trust region (using minimum_size), the contraction factor for the trust region size (using contraction_factor) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using expansion_factor) used when the surrogatemodel is performing well. Two additional commands are the trust region size contraction threshold (using contract_threshold) and the trust region size expansion threshold (using expand_threshold). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command contract_threshold sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command expand_threshold determines the trust region value above which the trust region will expand for the next SBL iteration.

contract_threshold

- Keywords Area
- methodology
- surrogate_based_local
- trust_region
- contract_threshold

Shrink trust region if trust region ratio is below this value

Specification

Alias: none
Argument(s): REAL
Default: 0.25
expand_threshold

- Keywords Area
- method
- surrogate_based_local
- trust_region
- expand_threshold

Expand trust region if trust region ratio is above this value

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 0.75

**Description**

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 0.25
6.2. METHOD

Description

For pattern search methods, \texttt{contraction\_factor} specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is \(1/\texttt{contraction\_factor}\)

\texttt{expansion\_factor}

- Keywords Area
- method
- surrogate\_based\_local
- trust\_region
- expansion\_factor

Trust region expansion factor

Specification

Alias: none
Argument(s): REAL
Default: 2.0

Description

The \texttt{trust\_region} optional group specification can be used to specify the initial size of the trust region (using \texttt{initial\_size}) relative to the total variable bounds, the minimum size of the trust region (using \texttt{minimum\_size}), the contraction factor for the trust region size (using \texttt{contraction\_factor}) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using \texttt{expansion\_factor}) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using \texttt{contract\_threshold}) and the trust region size expansion threshold (using \texttt{expand\_threshold}). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command \texttt{contract\_threshold} sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command \texttt{expand\_threshold} determines the trust region value above which the trust region will expand for the next SBL iteration.

\texttt{approx\_subproblem}

- Keywords Area
- method
- surrogate\_based\_local
- approx\_subproblem

Identify functions to be included in surrogate merit function
Specification

Alias: none

Argument(s): none

Default: original, primary, original, constraints
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>objective formulation (Group 1)</td>
<td>original_primary</td>
<td>Construct approximations of all primary functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>single_objective</td>
<td>Construct approximation a single objective functions only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>augmented_lagrangian_objective</td>
<td>Augmented Lagrangian approximate subproblem formulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lagrangian_objective</td>
<td>Lagrangian approximate subproblem formulation</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>constraint formulation (Group 2)</td>
<td>original_constraints</td>
<td>Use the constraints directly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linearized_constraints</td>
<td>Use linearized approximations to the constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_constraints</td>
<td>Don’t use constraints</td>
</tr>
</tbody>
</table>

Description

First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (original_primary), a single objective function (single_objective) formed from the primary function surrogates, or either an augmented Lagrangian merit function (augmented_lagrangian_objective) or a Lagrangian merit function (lagrangian_objective) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition. Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (original_constraints) or linearized approximations to the surrogate constraints (linearized_constraints), or constraints can be omitted from the subproblem (no_constraints).

original_primary
- Keywords Area
- method
- surrogate_based_local
- approx_subproblem
- original_primary
Construct approximations of all primary functions

**Specification**

Alias: none  
Argument(s): none

**Description**

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in[23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual[4]. First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (original primary), a single objective function (single objective) formed from the primary function surrogates, or either an augmented Lagrangian merit function (augmented lagrangian objective) or a Lagrangian merit function (lagrangian objective) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.

**single_objective**

- Keywords Area
- method
- surrogate based local
- approx subproblem
- single objective

Construct approximation a single objective functions only

**Specification**

Alias: none  
Argument(s): none

**Description**

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in[23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual[4]. First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (original primary), a single objective function (single objective) formed from the primary function surrogates, or either an augmented Lagrangian merit function (augmented lagrangian objective) or a Lagrangian merit function (lagrangian objective) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.
6.2. METHOD

augmented_lagrangian_objective

- Keywords Area
- method
- surrogate_based_local
- approx_subproblem
- augmented_lagrangian_objective

Augmented Lagrangian approximate subproblem formulation

Specification

Alias: none
Argument(s): none

Description

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in [23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual [4]. First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (original_primary), a single objective function (single_objective) formed from the primary function surrogates, or either an augmented Lagrangian merit function (augmented_lagrangian_objective) or a Lagrangian merit function (lagrangian_objective) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.

lagrangian_objective

- Keywords Area
- method
- surrogate_based_local
- approx_subproblem
- lagrangian_objective

Lagrangian approximate subproblem formulation

Specification

Alias: none
Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in[23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual[4]. First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (original_primary), a single objective function (single_objective) formed from the primary function surrogates, or either an augmented Lagrangian merit function (augmented_lagrangian_objective) or a Lagrangian merit function (lagrangian_objective) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.

original_constraints

- Keywords Area
- method
- surrogate_based_local
- approx_subproblem
- original_constraints

Use the constraints directly

Specification

Alias: none

Argument(s): none

Description

The surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (original_constraints) or linearized approximations to the surrogate constraints (linearized_constraints), or constraints can be omitted from the subproblem (no_constraints).

linearized_constraints

- Keywords Area
- method
- surrogate_based_local
- approx_subproblem
- linearized_constraints

Use linearized approximations to the constraints
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

The surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints \(\text{original}\_\text{constraints}\) or linearized approximations to the surrogate constraints \(\text{linearized}\_\text{constraints}\), or constraints can be omitted from the subproblem \(\text{no}\_\text{constraints}\).

no_constraints

- Keywords Area
- method
- surrogate_based_local
- approx_subproblem
- no_constraints

Don’t use constraints

Specification

Alias: none
Argument(s): none

Description

The surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints \(\text{original}\_\text{constraints}\) or linearized approximations to the surrogate constraints \(\text{linearized}\_\text{constraints}\), or constraints can be omitted from the subproblem \(\text{no}\_\text{constraints}\).

merit_function

- Keywords Area
- method
- surrogate_based_local
- merit_function

Select type of penalty or merit function

Specification

Alias: none
Argument(s): none

Default: augmented_lagrangian_merit
### Description

Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by surrogate-based local iteration number (penalty_merit), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (adaptive_penalty_merit), a Lagrangian merit function which employs first-order Lagrange multiplier updates (lagrangian_merit), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (augmented_lagrangian_merit). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [16].

**penalty_merit**
- Keywords Area
- method
- surrogate_based_local
- merit_function
- penalty_merit
  - Use penalty merit function

### Specification

**Alias:** none

**Argument(s):** none

### Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (original_constraints) or linearized approximations to the surrogate constraints (linearized_constraints), or constraints can be omitted from the subproblem (no_constraints). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be
6.2. METHOD

Selected to be a simple penalty function with penalty ramped by SBL iteration number (penalty_merit), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (adaptive_penalty_merit), a Lagrangian merit function which employs first-order Lagrange multiplier updates (lagrangian_merit), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (augmented_lagrangian_merit). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [16].

adaptive_penalty_merit

- Keywords Area
- method
- surrogate_based_local
- merit_function
- adaptive_penalty_merit

Use adaptive penalty merit function

Specification

Alias: none

Argument(s): none

Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (original_constraints) or linearized approximations to the surrogate constraints (linearized_constraints), or constraints can be omitted from the subproblem (no_constraints). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by SBL iteration number (penalty_merit), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (adaptive_penalty_merit), a Lagrangian merit function which employs first-order Lagrange multiplier updates (lagrangian_merit), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (augmented_lagrangian_merit). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [16].

lagrangian_merit

- Keywords Area
- method
- surrogate_based_local
- merit_function
- lagrangian_merit

Use first-order Lagrangian merit function
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by SBL iteration number (`penalty_merit`), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (`adaptive_penalty_merit`), a Lagrangian merit function which employs first-order Lagrange multiplier updates (`lagrangian_merit`), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (`augmented_lagrangian_merit`). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in[16].

augmented_lagrangian_merit

- Keywords Area
- method
- surrogate_based_local
- merit_function
- augmented_lagrangian_merit

Use combined penalty and zeroth-order Lagrangian merit function

Specification

Alias: none

Argument(s): none

Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by SBL iteration number (`penalty_merit`), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (`adaptive_penalty_merit`), a Lagrangian merit function which employs first-order Lagrange multiplier updates (`lagrangian_merit`), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (`augmented_lagrangian_merit`). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in[16].
6.2. METHOD

**acceptance_logic**
- Keywords Area
- method
- surrogate_based_local
- acceptance_logic

Set criteria for trusted surrogate

**Specification**

**Alias:** none
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group acceptance logic (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>tr_ratio</td>
<td>Surrogate-Based Local iterate acceptance logic</td>
</tr>
</tbody>
</table>

**Description**

Following calculation of the merit function for the new iterate, the iterate is accepted or rejected and the trust region size is adjusted for the next surrogate_based_local iteration. Iterate acceptance is governed either by a trust region ratio \((tr\_ratio)\) formed from the merit function values or by a filter method \((\text{filter})\); however, trust region resizing logic is currently based only on the trust region ratio. For infeasible iterates, constraint relaxation can be used for balancing constraint satisfaction and progress made toward an optimum.

**tr_ratio**
- Keywords Area
- method
- surrogate_based_local
- acceptance_logic
- tr_ratio

Surrogate-Based Local iterate acceptance logic

**Specification**

**Alias:** none
**Argument(s):** none
Description

Following calculation of the merit function for the new iterate, the iterate is accepted or rejected and the trust region size is adjusted for the next SBL iteration. Iterate acceptance is governed either by a trust region ratio ($tr\_ratio$) formed from the merit function values or by a filter method ($filter$); however, trust region resizing logic is currently based only on the trust region ratio. For infeasible iterates, constraint relaxation can be used for balancing constraint satisfaction and progress made toward an optimum. The command $constraint\_relax$ followed by a method name specifies the type of relaxation to be used. Currently, homotopy\[68\] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

filter

- Keywords Area
- method
- surrogate\_based\_local
- acceptance\_logic
- filter

Surrogate-Based Local iterate acceptance logic

Specification

Alias: none
Argument(s): none

Description

Following calculation of the merit function for the new iterate, the iterate is accepted or rejected and the trust region size is adjusted for the next SBL iteration. Iterate acceptance is governed either by a trust region ratio ($tr\_ratio$) formed from the merit function values or by a filter method ($filter$); however, trust region resizing logic is currently based only on the trust region ratio. For infeasible iterates, constraint relaxation can be used for balancing constraint satisfaction and progress made toward an optimum. The command $constraint\_relax$ followed by a method name specifies the type of relaxation to be used. Currently, homotopy\[68\] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

constraint\_relax

- Keywords Area
- method
- surrogate\_based\_local
- constraint\_relax

Enable constraint relaxation
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: no relaxation

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>homotopy</td>
<td>Surrogate-Based local constraint relaxation method for infeasible iterates</td>
</tr>
</tbody>
</table>

Description

The command `constraint_relax` followed by a method name specifies the type of relaxation to be used. Currently, `homotopy` [68] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

**homotopy**

- **Keywords Area**
- **method**
- **surrogate_based_local**
- **constraint_relax**
- **homotopy**

Surrogate-Based local constraint relaxation method for infeasible iterates

Specification

Alias: none
Argument(s): none

Description

Currently, `homotopy`[68] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

**max_iterations**

- **Keywords Area**
- **method**
- **surrogate_based_local**
- **max_iterations**

Stopping criterion based on number of iterations
CHAPTER 6. KEYWORDS AREA

**Topics**

This keyword is related to the topics:

- method
- independent
- controls

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-
_global: 25+n)

**Description**

The maximum number of iterations (default: 100). See also max_function_evaluations.

**convergence_tolerance**

- Keywords Area
- method
- surrogate_based_local
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- method
- independent
- controls

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 1.e-4

**Description**

The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.
Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

constraint_tolerance

- Keywords Area
- method
- surrogate_based_local
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method-independent_controls

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
• NPSOL: dependent upon the machine precision, typically on the order of $1.e^{-8}$ for double precision computations

### 6.2.9 surrogate_based_global

- **Keywords Area**
- **method**
- **surrogate_based_global**

Global Surrogate Based Optimization

**Topics**

This keyword is related to the topics:

- **surrogate_based_optimization_methods**

**Specification**

Alias: none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong></td>
<td></td>
<td>method_pointer</td>
<td>Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td>method_name</td>
<td>Specify sub-method by name</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
<td>replace_points</td>
<td>(Recommended) Replace points in the surrogate training set, instead of appending</td>
</tr>
</tbody>
</table>
6.2. METHOD

**Optional**

<table>
<thead>
<tr>
<th><strong>max_iterations</strong></th>
<th>Stopping criterion based on number of iterations</th>
</tr>
</thead>
</table>

**Description**

The `surrogate_based_global` specification must identify:

- a sub-method, using either `method_pointer` or `method_name`
- `model_pointer` must be used to identify a surrogate model

The `surrogate_based_global` works in an iterative scheme where optimization is performed on a global surrogate using the same bounds during each iteration.

- In one iteration, the optimal solutions of the surrogate model are found, and then a selected set of these optimal surrogate solutions are passed to the next iteration.
- At the next iteration, these surrogate points are evaluated with the ”truth” model, and then these points are added back to the set of points upon which the next surrogate is constructed.

In this way, the optimization acts on a more accurate surrogate during each iteration, presumably driving to optimality quickly.

**Method Independent Controls**

- `max_iterations` is used as a stopping criterion (see note below)

**Notes**

We have some cautionary notes before using the surrogate-based global method:

- **This approach has no guarantee of convergence.**
- One might first try a single minimization method coupled with a surrogate model prior to using the surrogate-based global method. This is essentially equivalent to setting `max_iterations` to 1 and will allow one to get a sense of what surrogate types are the most accurate to use for the problem.
- Also note that one can specify that surrogates be built for all primary functions and constraints or for only a subset of these functions and constraints. This allows one to use a ”truth” model directly for some of the response functions, perhaps due to them being much less expensive than other functions.
- We initially recommend a small number of maximum iterations, such as 3-5, to get a sense of how the optimization is evolving as the surrogate gets updated. If it appears to be changing significantly, then a larger number (used in combination with restart) may be needed.

**Theory**

In surrogate-based optimization (SBO) and surrogate-based nonlinear least squares (SBNLS), minimization occurs using a set of one or more approximations, defined from a surrogate model, that are built and periodically updated using data from a ”truth” model. The surrogate model can be a global data fit (e.g., regression or interpolation of data generated from a design of computer experiments), a multipoint approximation, a local Taylor Series expansion, or a model hierarchy approximation (e.g., a low-fidelity simulation model), whereas the truth model involves a high-fidelity simulation model. The goals of surrogate-based methods are to reduce the total
number of truth model simulations and, in the case of global data fit surrogates, to smooth noisy data with an
easily navigated analytic function.

It was originally designed for MOGA (a multi-objective genetic algorithm). Since genetic algorithms often
need thousands or tens of thousands of points to produce optimal or near-optimal solutions, the use of surrogates
can be helpful for reducing the truth model evaluations. Instead of creating one set of surrogates for the individual
objectives and running the optimization algorithm on the surrogate once, the idea is to select points along the
(surrogate) Pareto frontier, which can be used to supplement the existing points.

In this way, one does not need to use many points initially to get a very accurate surrogate. The surrogate
becomes more accurate as the iterations progress.

**See Also**

These keywords may also be of interest:

- efficient_global
- surrogate_based_local

**method_pointer**

- Keywords Area
- method
- surrogate_based_global
- method_pointer

Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem

**Topics**

This keyword is related to the topics:

- block_pointer

**Specification**

Alias: approx_method_pointer

*Argument(s):* STRING

**Description**

The `method_pointer` keyword is used to specify a pointer to an optimization or least-squares sub-method to
apply in the context of what could be described as hierarchical methods. In surrogate-based methods, the sub-
method is applied to the surrogate model. In the branch-and-bound method, the sub-method is applied to the
relaxed sub-problems.

Any `model_pointer` identified in the sub-method specification is ignored. Instead, the parent method is
responsible for selecting the appropriate model to use as specified by its `model_pointer`. In surrogate-based
methods, it is a surrogate model defined using its `model_pointer`. In branch-and-bound methods, it is the
relaxed model that is constructed internally from the original model.
method_name

- Keywords Area
- method
- surrogate_based_global
- method_name

Specify sub-method by name

Specification

Alias: approx_method_name
Argument(s): STRING

Description

The method_name keyword is used to specify a sub-method by Dakota method name (e.g. ‘npsol_sqp’) rather than block pointer. The method will be executed using its default settings. The optional model_pointer specification can be used to associate a model block with the method.

model_pointer

- Keywords Area
- method
- surrogate_based_global
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: approx_model_pointer
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)
**Description**

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURRE'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
      0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURRE'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
```
6.2. METHOD

```
response_functions = 3
no_gradients
no_hessians
```

**replace_points**

- **Keywords Area**
- **method**
- **surrogate_based_global**
- **replace_points**

(Recommended) Replace points in the surrogate training set, instead of appending

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** Points appended, not replaced

**Description**

The user has the option of appending the optimal points from the surrogate model to the current set of truth points or using the optimal points from the surrogate model to replace the optimal set of points from the previous iteration. Although appending to the set is the default behavior, at this time we strongly recommend using the option `replace_points` because it appears to be more accurate and robust.

**max_iterations**

- **Keywords Area**
- **method**
- **surrogate_based_global**
- **max_iterations**

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)
Description

The maximum number of iterations (default: 100). See also \texttt{max\_function\_evaluations}.

6.2.10 \texttt{dot\_frcg}

- \textbf{Keywords Area}
- \textbf{method}
- \textbf{dot\_frcg}

A conjugate gradient optimization method

Specification

\textbf{Alias:} none

\textbf{Argument(s):} none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>\texttt{max_iterations}</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>\texttt{convergence_tolerance}</td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td>\texttt{constraint_tolerance}</td>
<td>constraint_tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td>Optional</td>
<td>\texttt{speculative}</td>
<td>speculative</td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td>Optional</td>
<td>\texttt{max_function_evaluations}</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
</tbody>
</table>
### Optional

<table>
<thead>
<tr>
<th><strong>Optional</strong></th>
<th><strong>scaling</strong></th>
<th>Turn on scaling for variables, responses, and constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_inequality_.constraint_matrix</strong></td>
<td>Define coefficients of the linear inequality constraints</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_inequality_.lower_bounds</strong></td>
<td>Define lower bounds for the linear inequality constraint</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_inequality_.upper_bounds</strong></td>
<td>Define upper bounds for the linear inequality constraint</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_inequality_.scale_types</strong></td>
<td>Specify how each linear inequality constraint is scaled</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_inequality_.scales</strong></td>
<td>Define the characteristic values to scale linear inequalities</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_equality_.constraint_matrix</strong></td>
<td>Define coefficients of the linear equalities</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_equality_.targets</strong></td>
<td>Define target values for the linear equality constraints</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_equality_.scale_types</strong></td>
<td>Specify how each linear equality constraint is scaled</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>linear_equality_.scales</strong></td>
<td>Define the characteristic values to scale linear equalities</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td><strong>model_pointer</strong></td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

### Description

This is a duplicated keyword. Please use `dot` instead.

We here provide a caution regarding `dot_frcg`. In DOT Version 4.20, we have noticed inconsistent behavior of this algorithm across different versions of Linux. Our best assessment is that it is due to different treatments of uninitialized variables. As we do not know the intention of the code authors and maintaining DOT source code is outside of the Dakota project scope, we have not made nor are we recommending any code changes to address this. However, all users who use `dot_frcg` in DOT Version 4.20 should be aware that results may not be reliable.
CHAPTER 6. KEYWORDS AREA

See Also

These keywords may also be of interest:

- frcg

max_iterations

- Keywords Area
- method
- dot_frcg
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global {reliability, interval_est, evidence} / efficient_global: 25+n)

Description

The maximum number of iterations (default: 100). See also max.function_evaluations.

convergence_tolerance

- Keywords Area
- method
- dot_frcg
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls
6.2. METHOD

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration. Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

constraint_tolerance

- Keywords Area
  - method
  - dot_frcg
  - constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method_independent_controls
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): REAL
  Default: Library default

Description

The `constraint_tolerance` specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e^{-8}$ for double precision computations

speculative

- Keywords Area
- method
- dot_frcg
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
  Argument(s): none
  Default: no speculation
6.2. **METHOD**

**Description**

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- **Keywords Area**
  - method
  - dot_frcg
  - max_function_evaluations

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): INTEGER
Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling

- Keywords Area
- method
- dot_frcg
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none
Argument(s): none
Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values
6.2. METHOD

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   • two-sided bounds scaled into the interval [0,1];
   • one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   • no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling
   First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.
   Logarithmic scaling is not available for linear constraints.
   When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

• none, auto, log - optional
• value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
linear_inequality_constraint_matrix

- Keywords Area
- method
- dot_frcg
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

Description
In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_inequality_upper_bounds. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

linear_inequality_lower_bounds

- Keywords Area
- method
- dot_frcg
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics
This keyword is related to the topics:
- linear_constraints
6.2. METHOD

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = -infinity

Description

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[ a_l \leq Ax \leq a_u \]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[ Ax \leq 0.0 \]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \( +\text{bigRealBoundSize} \) (1.e+30, as defined in Minimizer) are treated as \( +\infty \) and any lower bound values less than \( -\text{bigRealBoundSize} \) are treated as \( -\infty \).

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \( -\text{DBL\_MAX} < -\text{bigRealBoundSize} \)).

linear_inequality_upper_bounds

- Keywords Area
- method
- dot_frcg
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.
CHAPTER 6. KEYWORDS AREA

Description

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
\begin{align*}
\begin{aligned}
a_l &\leq A x \\ A &\subseteq \text{constraint matrix of variable coefficients.}
\end{aligned}
\end{align*}
\]

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \(+\text{bigRealBoundSize} \ (1.e+30, \text{as defined in Minimizer})\) are treated as \(+\text{infinity}\) and any lower bound values less than \(-\text{bigRealBoundSize}\) are treated as \(-\text{infinity}\).

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\text{DBL\_MAX} < \text{-bigRealBoundSize}\)).

linear_inequality_scale_types

- Keywords Area
- method
- dot_frcg
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = ”none”

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ’none’ - no scaling
- ’value’ - characteristic value if this is chosen, then linear_inequality_scales must be specified
6.2. METHOD

• ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_i = \frac{x_i - x_O}{x_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**Linear_inequality_scales**

• Keywords Area
  • method
  • dot_frcg
  • linear_inequality_scales

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

• linear_constraints

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** vector values = 1. (no scaling)

**Description**

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.  
Behavior depends on the choice of linear_inequality_scale_type:

• scale_type - behavior of linear_inequality_scales

• ’none’ - ignored
• ‘value’ - required
• ‘auto’ - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables x:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i \left( \text{diag}(x_M)\tilde{x} + x_O \right) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear equality constraint matrix**

- Keywords Area
- method
- dot.frcg
- linear.equality.constraint_matrix

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- linear.constraints

**Specification**

*Alias:* none

*Argument(s):* REALLIST

*Default:* no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$
6.2. METHOD

linear_equalit_targets

- Keywords Area
- method
- dot_frcg
- linear_equalit_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0.0$$

linear_equalit_scale_types

- Keywords Area
- method
- dot_frcg
- linear_equalit_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

linear Equality_scales_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear equality scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear Equality_scales

- Keywords Area
- method
- dot_frcg
- linear Equality_scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear constraints
6.2. METHOD

Specification

Alias: none
Argument(s): REALIST
Default: vector values = 1. (no scaling)

Description

Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

model_pointer

- Keywords Area
- method
- dot_frcg
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYWORDS AREA

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
              0.1 0.2 0.6
              0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
```
6.2. METHOD

response_functions = 3
no_gradients
no_hessians

6.2.11 dot_mmfd

- Keywords Area
- method
- dot_mmfd

Method of feasible directions

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>constraint_-tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>speculative</td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_function_-evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
</tbody>
</table>
| Optional | scaling                  | Turn on scaling for variables, responses, and constraints
| Optional | linear_inequality_-constraint_matrix | Define coefficients of the linear inequality constraints
| Optional | linear_inequality_-lower_bounds | Define lower bounds for the linear inequality constraint
| Optional | linear_inequality_-upper_bounds | Define upper bounds for the linear inequality constraint
| Optional | linear_inequality_-scale_types | Specify how each linear inequality constraint is scaled
| Optional | linear_inequality_-scales | Define the characteristic values to scale linear inequalities
| Optional | linear_equation_-constraint_matrix | Define coefficients of the linear equalities
| Optional | linear_equation_-targets | Define target values for the linear equality constraints
| Optional | linear_equation_-scale_types | Specify how each linear equality constraint is scaled
| Optional | linear_equation_-scales | Define the characteristic values to scale linear equalities
| Optional | model_pointer | Identifier for model block to be used by a method

**Description**

This is a duplicated keyword. Please use dot instead.

**See Also**

These keywords may also be of interest:
6.2. METHOD

- mmfd

**max_iterations**

- Keywords Area
- method
- dot_mmfd
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also max_function_evaluations.

**convergence_tolerance**

- Keywords Area
- method
- dot_mmfd
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

**Argument(s):** REAL

**Default:** 1.e-4
Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do. Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

constraint_tolerance

- Keywords Area
- method
- dot_mmfd
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: Library default
Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied. If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e^{-8}$ for double precision computations

speculative

- Keywords Area
- method
- dot_mmfd
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- dot_mmfd
- max_function_evaluations

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

 Argument(s): INTEGER

 Default: 1000

**Description**

The maximum number of function evaluations (default: 1000). See also max_iterations.
6.2. METHOD

scaling

- Keywords Area
- method
- dot_mmfd
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and _scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a+scales keyword, which gives characteristic values
- a+scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of _scales keyword is ignored
2. value - multiplicative scaling
3. **auto** - automatic scaling  
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   
   - two-sided bounds scaled into the interval $[0,1]$;
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

4. **log** - logarithmic scaling  
   First, any characteristic values from the optional \texttt{\_scales} specification are applied. Then logarithm base 10 scaling is applied.

   Logarithmic scaling is not available for linear constraints.

   When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**  
The \texttt{\_scales} keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding \texttt{\_scale_type}, as described above.

   Depending on the scale type, the characteristic values may be required or optional.

   - \texttt{none}, \texttt{auto}, \texttt{log} - optional
   - \texttt{value} - required.

   A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10*DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**  
The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
6.2. METHOD

linear_inequality_constraint_matrix

- Keywords Area
- method
- dot_mmfd
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear inequality constraints

Description

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_inequality_upper_bounds. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

linear_inequality_lower_bounds

- Keywords Area
- method
- dot_mmfd
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): REALLIST
  Default: vector values = -infinity

Description

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \(+\)bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as \(+\)infinity and any lower bound values less than \(-\)bigRealBoundSize are treated as \(-\)infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\)DBL\_MAX < \(-\)bigRealBoundSize).

linear_inequality_upper_bounds

- Keywords Area
- method
- dot_mmfd
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none
  Argument(s): REALLIST
  Default: vector values = 0.
6.2. **METHOD**

**Description**

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \(+\text{bigRealBoundSize}\) (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than \(-\text{bigRealBoundSize}\) are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\text{DBL\_MAX} < -\text{bigRealBoundSize}\)).

**linear_inequality_scale_types**

- Keywords Area
- method
- dot_mmfd
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** vector values = "none"

**Description**

`linear_inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `none` - no scaling
- `value` - characteristic value if this is chosen, then `linear_inequality_scales` must be specified
• 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}_j = \frac{x_j - x_{jO}}{x_{jM}}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_{jO}) \leq a_U$$

$$a_L - A_i x_{jO} \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_{jO}$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_inequality_scales

- Keywords Area
- method
- dot_mmf

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
- 'none' - ignored
6.2. METHOD

- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

**linear_equality_constraint_matrix**

- Keywords Area
- method
- dot_mmfd
- linear_equality_constraint_matrix

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$
linear.equality_targets

- Keywords Area
- method
- dot_mmfd
- linear.equality_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0 .

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear.equality_scale_types

- Keywords Area
- method
- dot_mmfd
- linear.equality_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints
6.2. METHOD

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear_equality_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear.equality.scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M - x^j_O}
$$

we have the following system for linear equality constraints

$$
a_L \leq A_i x \leq a_U
$$

$$
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
$$

$$
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
$$

$$
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear.equality.scales

- Keywords Area
- method
- dot_mmfd
- linear.equality.scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REALIST

Default: vector values = 1 . (no scaling)

Description

Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_j = \frac{x_j - x_M}{x_O}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

model_pointer

- Keywords Area
- method
- dot_mmfd
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SRR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
distribution cumulative

model
  id_model = 'SRR'
  surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
```
response_functions = 3
no_gradients
no_hessians

6.2.12 dot_bfgs

- Keywords Area
- method
- dot_bfgs

A conjugate gradient optimization method

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>constraint_-tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>speculative</td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td></td>
<td></td>
<td>max_function_-evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>scaling</th>
<th>Turn on scaling for variables, responses, and constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>linear_inequality_.constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_.lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_.upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_.scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_.scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_.constraint_matrix</td>
<td>Define coefficients of the linear equalities</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_.targets</td>
<td>Define target values for the linear equality constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_.scale_types</td>
<td>Specify how each linear equality constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_.scales</td>
<td>Define the characteristic values to scale linear equalities</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

Description

This is a duplicated keyword. Please use dot instead.

See Also

These keywords may also be of interest:
CHAPTER 6. KEYWORDS AREA

- **bfgs**

**max_iterations**

- **Keywords Area**
- **method**
- **dot_bfgs**
- **max_iterations**

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also **max_function_evaluations**.

**convergence_tolerance**

- **Keywords Area**
- **method**
- **dot_bfgs**
- **convergence_tolerance**

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 1.e-4
6.2. METHOD

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

constraint_tolerance

- Keywords Area
- method
- dot_bfgs
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: Library default
CHAPTER 6. KEYWORDS AREA

Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of 1.e-8 for double precision computations

speculative

- Keywords Area
- method
- dot_bfgs
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
Argument(s): none
Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

`max_function_evaluations`

- **Keywords Area**

- **method**

- **dot_bfgs**

- **max_function_evaluations**

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias**: none

**Argument(s)**: INTEGER

**Default**: 1000

**Description**

The maximum number of function evaluations (default: 1000). See also `max_iterations`. 
scaling

- **Keywords Area**
- **method**
- **dot_bfgs**
- **scaling**

Turn on scaling for variables, responses, and constraints

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no scaling

**Description**

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the `method`, `variables`, and `responses` blocks. When the `scaling` keyword is omitted, all `scale_types` and `scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `scales` keyword, which gives characteristic values
- a `scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. `none` (default) - no scaling, value of `scales` keyword is ignored
2. `value` - multiplicative scaling
6.2. METHOD

3. **auto** - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. **log** - logarithmic scaling

First, any characteristic values from the optional **scales** specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The **scales** keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding **scale_type**, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- **none**, **auto**, **log** - optional
- **value** - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0 \times 10^{-DBL_MIN}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
linear_inequality_constraint_matrix

- Keywords Area
- method
- dot_bfgs
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

Description
In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_inequality_upper_bounds. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

linear_inequality_lower_bounds

- Keywords Area
- method
- dot_bfgs
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics
This keyword is related to the topics:

- linear_constraints
Specification

Alias: none
Argument(s): REALLIST
Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$ a_l \leq Ax \leq a_u $$

Where $A$ is the constraint matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$ Ax \leq 0.0 $$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL}_{\text{MAX}} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- dot_bfgs
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none
Argument(s): REALLIST
Default: vector values = 0.
Description
In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \(+bigRealBoundSize\) (1.e+30, as defined in Minimizer) are treated as \(+\infty\) and any lower bound values less than \(-bigRealBoundSize\) are treated as \(-\infty\).

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-DBL_MAX < -bigRealBoundSize\)).

linear_inequality_scale_types

- **Keywords Area**
- **method**
- **dot_bfgs**
- **linear_inequality_scale_types**

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

- **linear_constraints**

Specification

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** vector values = "none"

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ‘none’ - no scaling
- ‘value’ - characteristic value if this is chosen, then linear_inequality_scales must be specified
6.2. METHOD

• ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

**linear_inequality_scales**

- Keywords Area
- method
- dot_bfgs
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

**Description**

Each entry in **linear_inequality_scales** may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of **linear_inequality_scale_type**:

- scale_type - behavior of **linear_inequality_scales**
- ‘none’ - ignored
CHAPTER 6. KEYWORDS AREA

- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

**linear_equality_constraint_matrix**

- Keywords Area
- method
- dot_bfgs
- linear_equality_constraint_matrix

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$
6.2. METHOD

linear_equal targets
• Keywords Area
• method
• dot_bfgs
• linear_equal_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:
• linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:
\[ Ax = a_t \]

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equal_scale_types
• Keywords Area
• method
• dot_bfgs
• linear_equal_scale_types

Specify how each linear equality constraint is scaled

Topics
This keyword is related to the topics:
• linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

`linear.equality.scale_types` provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then `linear.equality.scales` must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \hat{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

`linear.equality.scales`

- Keywords Area
- method
- dot_bfgs
- `linear.equality.scales`

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- `linear.constraints`
6.2. METHOD

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

Description

Each entry in `linear.equality.scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the `method` section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}_j = \frac{x_j - x_O}{x_M}
$$

we have the following system for linear inequality constraints

$$
a_L \leq A_i x \leq a_U
$$

$$
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
$$

$$
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
$$

$$
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

model_pointer

- Keywords Area
- method
- dot_bfgs
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYWORDS AREA

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

evironment
    tabular_graphics_data
    method_pointer = 'UQ'

method
    id_method = 'UQ'
    model_pointer = 'SURR'
    sampling,
        samples = 10
        seed = 98765 rng rnum2
        response_levels = 0.1 0.2 0.6
            0.1 0.2 0.6
            0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
    id_model = 'SURR'
        surrogate global,
        dace_method_pointer = 'DACE'
        polynomial quadratic

method
    id_method = 'DACE'
    model_pointer = 'DACE_M'
    sampling sample_type lhs
        samples = 121 seed = 5034 rng rnum2

model
    id_model = 'DACE_M'
        single
        interface_pointer = 'I1'

variables
    uniform_uncertain = 2
        lower_bounds = 0. 0.
        upper_bounds = 1. 1.
        descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
        system asynch evaluation_concurrency = 5
        analysis_driver = 'text_book'

responses
### 6.2. METHOD

response_functions = 3
no_gradients
no_hessians

#### 6.2.13 dot_slp

- **Keywords Area**
- **method**
- **dot_slp**

Sequential Linear Programming

**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>constraint_-tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>speculative</td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_function_-evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
</tbody>
</table>
### Description

This is a duplicated keyword. Please use `dot` instead.

### See Also

These keywords may also be of interest:

<table>
<thead>
<tr>
<th>Optional</th>
<th>scaling</th>
<th>Turn on scaling for variables, responses, and constraints. Define coefficients of the linear inequality constraints. Define lower bounds for the linear inequality constraint. Define upper bounds for the linear inequality constraint. Specify how each linear inequality constraint is scaled. Define the characteristic values to scale linear inequalities. Define target values for the linear equality constraints. Specify how each linear equality constraint is scaled. Define the characteristic values to scale linear equalities. Identifier for model block to be used by a method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>linear_inequality_-constraint_matrix</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-lower_bounds</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-upper_bounds</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scale_types</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scales</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-constraint_matrix</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-targets</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-scale_types</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-scales</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

- slp

max_iterations

- Keywords Area
- method
- dot_slp
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- dot_slp
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4
CHAPTER 6. KEYWORDS AREA

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

constraint_tolerance

- Keywords Area
- method
- dot_slp
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: Library default
6.2. METHOD

Description

The constraint tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e^{-8}$ for double precision computations

speculative

- Keywords Area
- method
- dot_slp
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method independent controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- dot_slp
- max_function_evaluations

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

**Argument(s):** INTEGER

**Default:** 1000

**Description**

The maximum number of function evaluations (default: 1000). See also max_iterations.
6.2. METHOD

scaling

- Keywords Area
- method
- dot_slp
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
  Argument(s): none
  Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. **auto** - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. **log** - logarithmic scaling

First, any characteristic values from the optional *scales* specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The *scales* keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type*, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- **none**, **auto**, **log** - optional
- **value** - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \( 1.0 \times 10^{-10} \). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
6.2. METHOD

linear_inequality_constraint_matrix

- **Keywords Area**
- **method**
- **dot_slp**
- **linear_inequality_constraint_matrix**

Define coefficients of the linear inequality constraints

**Topics**

This keyword is related to the topics:

- **linear_constraints**

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** no linear inequality constraints

**Description**

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$ a_l \leq Ax \leq a_u $$

Where the bounds are optionally specified by **linear_inequality_lower_bounds**, and **linear_inequality_upper_bounds**. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form $Ax \leq 0.0$.

linear_inequality_lower_bounds

- **Keywords Area**
- **method**
- **dot_slp**
- **linear_inequality_lower_bounds**

Define lower bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- **linear_constraints**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): REALLIST
  Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).

linear_inequality_upper_bounds

- Keywords Area
- method
- dot_slp
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none
  Argument(s): REALLIST
  Default: vector values = 0.
Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBLMAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

- Keywords Area
- method
- dot_slp
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_inequality_scales must be specified
• ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_j = \frac{x_j^O}{x_M^O}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_inequality_scales**

- Keywords Area
- method
- dot_slp
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

**Description**

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`
- ‘none’ - ignored
• ‘value’ - required
• ‘auto’ - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$: $\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$

we have the following system for linear inequality constraints

$a_L \leq A_i x \leq a_U$
$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$
$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear Equality Constraint Matrix**

- **Keywords** Area
- **method**
- **dot_slp**
- **linear equality constraint matrix**

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- **linear constraints**

**Specification**

- **Alias**: none
- **Argument(s)**: REALLIST
- **Default**: no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$Ax = a_t$
linear_equation_targets

- Keywords Area
- method
- dot_slp
- linear_equation_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equation_scale_types

- Keywords Area
- method
- dot_slp
- linear_equation_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints
6.2. METHOD

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

`linear.equality_scale_types` provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then `linear.equality_scales` must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M - x^j_O}
\]

we have the following system for linear equality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

`linear.equality_scales`

- Keywords Area
- method
- dot_slp
- `linear.equality_scales`

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- `linear_constraints`
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

Description

Each entry in `linearEquality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_j = \frac{x_j - x^O_j}{x^M_j}
\]

we have the following system for linear inequality constraints

\[
a_L \leq \tilde{A}_i \tilde{x} \leq a_U
\]

\[
a_L \leq \tilde{A}_i (\text{diag}(x_M) \tilde{x} + x^O) \leq a_U
\]

\[
a_L - A_i x^O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x^O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

model_pointer

- Keywords Area
- method
- dot_slp
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SRR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
          0.1 0.2 0.6
          0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SRR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
6.2.14  **dot_sqp**

- **Keywords Area**
- **method**
- **dot_sqp**

Sequential Quadratic Program

**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><strong>max_iterations</strong></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>convergence_tolerance</strong></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>constraint_tolerance</strong></td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>speculative</strong></td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>max_function_evaluations</strong></td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
</tbody>
</table>
### Description

This is a duplicated keyword. Please use `dot` instead.

### See Also

These keywords may also be of interest:
max_iterations

- Keywords Area
- method
- dot_sqp
- max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global {reliability , interval_est , evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- dot_sqp
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4
6.2. **METHOD**

**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative **convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- **DOT**: must be satisfied for two consecutive iterations
- **NPSOL**: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- **NL2SOL**: See `nl2sol`

**constraint_tolerance**

- **Keywords Area**
- **method**
- **dot_sqp**
- **constraint_tolerance**

The maximum allowable value of constraint violation still considered to be feasible

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias**: none

**Argument(s)**: REAL

**Default**: Library default
CHAPTER 6. KEYWORDS AREA

Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e-8$ for double precision computations

speculative

- Keywords Area
- method
- dot_sqp
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for
efficient utilization of multiple processors. While the total amount of work performed will generally increase
(since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run
time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were
already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative
specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries,
and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer
to responses gradient section for information on these specifications). It should not be selected with vendor
numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose.
In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support
speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or
analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information,
where applicable, for parallel optimization studies. By speculating that the derivative information at the current
point will be used later, the complete data set (all available gradient/Hessian information) can be computed on
every function evaluation. While some of these computations will be wasted, the positive effects are a consist-
ent parallel load balance and usually shorter wall clock time. The speculative specification is applicable
only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor
numerical gradients).

max_function_evaluations

- Keywords Area
- method
- dot_sqp
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.
Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:

• method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the `scaling` keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

• a *scales keyword, which gives characteristic values
• a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling

---

scaling

• Keywords Area
• method
• dot_sqp
• scaling

Topics
This keyword is related to the topics:

• method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the `scaling` keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

• a *scales keyword, which gives characteristic values
• a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling

First, any characteristic values from the optional *_scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10=DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
linear_inequality_constraint_matrix

- Keywords Area
- method
- dot_sqp
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

**Topics**
This keyword is related to the topics:
- linear_constraints

**Specification**
Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

**Description**
In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_inequality_upper_bounds. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

linear_inequality_lower_bounds

- Keywords Area
- method
- dot_sqp
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

**Topics**
This keyword is related to the topics:
- linear_constraints
6.2. METHOD

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ (1.0e+30, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL}MAX < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- dot_sqp
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.
Description

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).

linear_inequality_scale_types

- Keywords Area
- method
- dot_sqp
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_inequality_scales must be specified
6.2. METHOD

- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

linear_inequality_scales

- Keywords Area
- method
- dot_sqp
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
- 'none' - ignored
• 'value' - required
• 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear.equality.constraint_matrix**

- Keywords Area
- method
- dot.sqp
- linear.equality.constraint_matrix

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$
linear_equation_targets

- Keywords Area
- method
- dot_sqp
- linear_equation_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equation_scale_types

- Keywords Area
- method
- dot_sqp
- linear_equation_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

linear.equality_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear.equality_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear.equality_scales

- Keywords Area
- method
- dot.sqp
- linear.equality_scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear.constraints
6.2. METHOD

Specification

Alias: none

Argument(s): REALIST

Default: vector values = 1. (no scaling)

Description

Each entry in `linear_equality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the `method` section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

**model_pointer**

- **Keywords Area**
- **method**
- **dot_sqp**
- **model_pointer**

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- **block_pointer**

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYWORDS AREA

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```plaintext
environment
    tabular_graphics_data
    method_pointer = 'UQ'

method
    id_method = 'UQ'
    model_pointer = 'Surr'
    sampling,
        samples = 10
        seed = 98765 rng rnum2
        response_levels = 0.1 0.2 0.6
        0.1 0.2 0.6
        0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
    id_model = 'Surr'
    surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
    id_method = 'DACE'
    model_pointer = 'DACE_M'
    sampling sample_type lhs
        samples = 121 seed = 5034 rng rnum2

model
    id_model = 'DACE_M'
    single
    interface_pointer = 'I1'

variables
    uniform_uncertain = 2
        lower_bounds = 0. 0.
        upper_bounds = 1. 1.
        descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
```
6.2. METHOD

response_functions = 3
no_gradients
no_hessians

6.2.15 dot

- Keywords Area
- method
- dot

Access to methods in the DOT package

Topics
This keyword is related to the topics:
- package.dot

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Dakota Keyword</th>
<th>Description</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>frcg</td>
<td>Group 1</td>
<td>A conjugate gradient optimization method</td>
</tr>
<tr>
<td>mmfd</td>
<td>Method of feasible directions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bfgs</td>
<td>A conjugate gradient optimization method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slp</td>
<td>Sequential Linear Programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sqp</td>
<td>Sequential Quadratic Programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Compute speculative gradients</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
<td></td>
</tr>
</tbody>
</table>
### Description

The DOT library [85] contains nonlinear programming optimizers, specifically the Broyden-Fletcher-Goldfarb-Shanno (Dakota’s dot_bfgs method) and Fletcher-Reeves conjugate gradient (Dakota’s dot_frcg method) methods for unconstrained optimization, and the modified method of feasible directions (Dakota’s dot_mmfd method), sequential linear programming (Dakota’s dot_slp method), and sequential quadratic programming (Dakota’s dot_sqp method) methods for constrained optimization.

Specialized handling of linear constraints is supported with DOT; linear constraint coefficients, bounds, and targets can be provided to DOT at start-up and tracked internally.

One of the five available methods in **Group 1** must be specified.

All these methods take the same **Optional Keywords**, dealing with linear equality and inequality constraints.

**Method Independent Controls - Stopping Criteria**

Stopping criteria are set by:

- `max_iterations`
- `max_function_evaluations`
- `convergence_tolerance`
- `constraint_tolerance`

Note: The `convergence_tolerance` criterion must be satisfied for two consecutive iterations before DOT will terminate.

**Method Independent Controls - Output**

The output verbosity specification controls the amount of information generated by DOT: the `silent` and `quiet` settings result in header information, final results, and objective function, constraint, and parameter information on each iteration; whereas the `verbose` and `debug` settings add additional information on gradients, search direction, one-dimensional search results, and parameter scaling factors.

**Concurrency**

DOT contains no parallel algorithms which can directly take advantage of concurrent evaluations. However, if `numerical_gradients` with `method_source dakota` is specified, then the finite difference function evaluations can be performed concurrently (using any of the parallel modes described in the Users Manual[4]).

<table>
<thead>
<tr>
<th>Optional</th>
<th>linear_equality._constraint_matrix</th>
<th>Define coefficients of the linear equalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>linear_equality._targets</td>
<td>Define target values for the linear equality constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality._scale_types</td>
<td>Specify how each linear equality constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality._scales</td>
<td>Define the characteristic values to scale linear equalities</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>
In addition, if speculative is specified, then gradients (dakota numerical or analytic gradients) will be computed on each line search evaluation in order to balance the load and lower the total run time in parallel optimization studies.

**frcg**

- **Keywords Area**

- **method**

- **dot**

- **frcg**

A conjugate gradient optimization method

**Topics**

This keyword is related to the topics:

- **package_dot**

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

We here provide a caution regarding **dot_frcg**. In DOT Version 4.20, we have noticed inconsistent behavior of this algorithm across different versions of Linux. Our best assessment is that it is due to different treatments of uninitialized variables. As we do not know the intention of the code authors and maintaining DOT source code is outside of the Dakota project scope, we have not made nor are we recommending any code changes to address this. However, all users who use **dot_frcg** in DOT Version 4.20 should be aware that results may not be reliable.

See **package_dot** for information related to all **DOT** methods.

**See Also**

These keywords may also be of interest:

- **bfgs**

- **mmfd**

- **slp**

- **sqp**
6.2. METHOD

**mmfd**

- **Keywords Area**
- method
- dot
- mmfd

Method of feasible directions

**Topics**

This keyword is related to the topics:

- package dot

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See package dot for information related to all DOT methods.

**See Also**

These keywords may also be of interest:

- bfgs
- frcg
- slp
- sqp

**bfgs**

- **Keywords Area**
- method
- dot
- bfgs

A conjugate gradient optimization method

**Topics**

This keyword is related to the topics:

- package dot
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): none

Description

See package dot for information related to all DOT methods.

See Also

These keywords may also be of interest:

- frcg
- mmfd
- slp
- sqp

slp

- Keywords Area
- method
- dot
- slp

Sequential Linear Programming

Topics

This keyword is related to the topics:

- package dot

Specification

Alias: none
  Argument(s): none

Description

See package dot for information related to all DOT methods.
See Also

These keywords may also be of interest:

- bfgs
- frcg
- mmfd
- sqp

sqp

- Keywords Area
- method
- dot
- sqp

Sequential Quadratic Program

Topics

This keyword is related to the topics:

- package_dot

Specification

Alias: none

Argument(s): none

Description

See package_dot for information related to all DOT methods.

See Also

These keywords may also be of interest:

- bfgs
- frcg
- mmfd
- slp
**max_iterations**

- **Keywords Area**
- **method**
- **dot**
- **max_iterations**

Stopping criterion based on number of iterations

### Topics
This keyword is related to the topics:

- **method_independent_controls**

### Specification

**Alias:** none  

**Argument(s):** INTEGER  

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence}/efficient-global: 25*n)

### Description
The maximum number of iterations (default: 100). See also **max.function_evaluations**.

**convergence_tolerance**

- **Keywords Area**
- **method**
- **dot**
- **convergence_tolerance**

Stopping criterion based on convergence of the objective function or statistics

### Topics
This keyword is related to the topics:

- **method_independent_controls**

### Specification

**Alias:** none  

**Argument(s):** REAL  

**Default:** 1.e-4
6.2. METHOD

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

**constraint_tolerance**

- **Keywords Area**
- method
- dot
- `constraint_tolerance`

The maximum allowable value of constraint violation still considered to be feasible

**Topics**

This keyword is related to the topics:

- `method-independent_controls`

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** Library default
CHAPTER 6. KEYWORDS AREA

Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of 1.e-8 for double precision computations

speculative

- Keywords Area
- method
- dot
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

max_function_evaluations

- Keywords Area
- method
- dot
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.
Keywords Area

method

dot

scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method
- independent_controls

Specification

Alias: none
- Argument(s): none
- Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10*DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```
linear_inequality_constraint_matrix

- Keywords Area
- method
- dot
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

Description
In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_inequality_upper_bounds. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

linear_inequality_lower_bounds

- Keywords Area
- method
- dot
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics
This keyword is related to the topics:
- linear_constraints
6.2. METHOD

Specification

Alias: none

Argument(s): REALLIST
Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $\text{bigRealBoundSize}$ (1.e+30, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- dot
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST
Default: vector values = 0.
CHAPTER 6. KEYWORDS AREA

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective.functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

- Keywords Area
- method
- dot

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = ”none”

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ’none’ - no scaling
- ’value’ - characteristic value if this is chosen, then linear_inequality_scales must be specified
6.2. METHOD

- ‘auto’ - automatic scaling. If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_0}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_0) \leq a_U$$

$$a_L - A_i x_0 \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_0$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_inequality_scales**

- Keywords Area
- method
- dot
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = 1. (no scaling)

**Description**

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`
- ‘none’ - ignored
• ‘value’ - required
• ‘auto’ - optional
If a single real value is specified it will apply to all components of the constraint.
Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

**linear_equality_constraint_matrix**

- Keywords Area
- method
- dot
- linear_equality_constraint_matrix

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

*Alias:* none
*Argument(s):* REALLIST
*Default:* no linear equality constraints

**Description**

In the equality case, the constraint matrix \( A \) provides coefficients for the variables on the left hand side of:

\[
Ax = a_t
\]
linear_equality_targets

- Keywords Area
- method
- dot
- linear_equality_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0.0$$

linear_equality_scale_types

- Keywords Area
- method
- dot
- linear_equality_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

linear Equality_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.
An entry may be selected for each constraint. The options are:

- ’none’ - no scaling
- ’value’ - characteristic value if this is chosen, then linear equality scales must be specified
- ’auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables x:

\[ \tilde{x}^j = \frac{x^j - x^j_O}{x^j_M} \]

we have the following system for linear equality constraints

\[ a_L \leq A_i x \leq a_U \]
\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear equality scales

- Keywords Area
- method
- dot
- linear equality scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear constraints
6.2. **METHOD**

**Specification**

**Alias:** none

**Argument(s):** REALIST

**Default:** vector values = 1. (no scaling)

**Description**

Each entry in `linear_equality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the **method** section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
$$

we have the following system for linear inequality constraints

$$
\begin{align*}
a_L & \leq A_i x \leq a_U \\
a_L & \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O & \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \\
\tilde{a}_L & \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\end{align*}
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

**model_pointer**

- **Keywords Area**
- **method**
- **dot**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)
Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
```
6.2. METHOD

response_functions = 3
no_gradients
no_hessians

6.2.16 conmin_freg

- Keywords Area
- method
- conmin_freg

A conjugate gradient optimization method

**Specification**

**Alias:** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>constraint_-tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>speculative</td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_function_-evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------</td>
<td>----------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_-constraint_matrix</td>
<td>Define coefficients of the linear equalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_-targets</td>
<td>Define target values for the linear equality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_-scale_types</td>
<td>Specify how each linear equality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_-scales</td>
<td>Define the characteristic values to scale linear equalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

This is a duplicated keyword. Please use conmin instead.

**See Also**

These keywords may also be of interest:
6.2. METHOD

- frcg

**max_iterations**

- Keywords Area
- method
- conmin_frcg
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global {reliability, interval_est, evidence} / efficient_global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also max_function_evaluations.

**convergence_tolerance**

- Keywords Area
- method
- conmin_frcg
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 1.e-4
Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

constraint_tolerance

- Keywords Area
- method
- conmin_frcg
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL
Default: Library default
6.2. METHOD

Description

The `constraint_tolerance` specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.0 \times 10^{-8}$ for double precision computations

speculative

- Keywords Area
- method
- conmin_frcg
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose.

In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- conmin_frcg
- max_function_evaluations

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

Argument(s): INTEGER

Default: 1000

**Description**

The maximum number of function evaluations (default: 1000). See also max_iterations.
6.2. METHOD

scaling

- Keywords Area
- method
- conmin_frcg
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
- Argument(s): none
- Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the `scaling` keyword is omitted, all `scale_types` and `*scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `*scales` keyword, which gives characteristic values
- a `*scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of `*scales` keyword is ignored
2. value - multiplicative scaling
3. **auto** - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

4. **log** - logarithmic scaling

First, any characteristic values from the optional `*scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_type`, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10*DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
6.2. **METHOD**

linear_inequality_constraint_matrix
- **Keywords Area**
- **method**
- **conmin_frcg**
- **linear_inequality_constraint_matrix**

Define coefficients of the linear inequality constraints

**Topics**
This keyword is related to the topics:
- **linear_constraints**

**Specification**

**Alias:** none
- **Argument(s):** REALLIST
- **Default:** no linear inequality constraints

**Description**
In the inequality case, the constraint matrix \( A \) provides coefficients for the variables in the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where the bounds are optionally specified by `linear_inequality_lower_bounds` and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

\[
Ax \leq 0.0
\]

linear_inequality_lower_bounds
- **Keywords Area**
- **method**
- **conmin_frcg**
- **linear_inequality_lower_bounds**

Define lower bounds for the linear inequality constraint

**Topics**
This keyword is related to the topics:
- **linear_constraints**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REALLIST
Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

\[ a_l \leq Ax \leq a_u \]

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[ Ax \leq 0.0 \]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- conmin_frcg
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST
Default: vector values = 0.
Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBLMAX} < -\text{bigRealBoundSize}$).

**linear_inequality_scale_types**

- Keywords Area
- method
- conmin_frcg
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

- linear_constraints

Specification

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** vector values = "none"

Description

`linear_inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear_inequality_scales` must be specified
• 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

linear_inequality_scales

• Keywords Area

• method

• conmin_frcg

• linear_inequality_scales

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

• linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

• scale_type - behavior of linear_inequality_scales

• 'none' - ignored
6.2. METHOD

- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear.equality.constraint_matrix**

- Keywords Area
- method
- conmin_frcg
- linear.equality.constraint_matrix

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$
linear_equality_targets

- Keywords Area
- method
- conmin_frcg
- linear_equality_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equality_scale_types

- Keywords Area
- method
- conmin_frcg
- linear_equality_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints
## 6.2. METHOD

### Specification

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** vector values = "none"

### Description

The `linear.equality.scale_types` provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear.equality.scales` must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied *after* any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M - x^j_O}
\]

we have the following system for linear equality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

### linear.equality.scales

- **Keywords** Area
- **method**
- **conmin_frcg**
- **linear.equality.scales**

Define the characteristic values to scale linear equalities

### Topics

This keyword is related to the topics:

- **linear.constraints**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REALIST

Default: vector values = 1 . (no scaling)

Description

Each entry in `linear.equality.scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}
\]

we have the following system for linear inequality constraints

\[
a_L \leq \tilde{A}_i \tilde{x} \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

**model_pointer**

- Keywords Area
- method
- conmin.frcg
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURRE'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynchronous evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
```
response_functions = 3
no_gradients
no_hessians

6.2.17 conmin_mfd

- Keywords Area
- method
- conmin_mfd

Method of feasible directions

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_tolerance</td>
<td></td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td></td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td></td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
</tbody>
</table>
### Description

This is a duplicated keyword. Please use `conmin` instead.

### See Also

These keywords may also be of interest:
max_iterations

• Keywords Area
• method
• conmin_mfd
• max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:

• method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

• Keywords Area
• method
• conmin_mfd
• convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:

• method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: 1.e-4
Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

Constraint Tolerance

- Keywords Area
- method
- conmin_mfd
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: Library default
CHAPTER 6. KEYWORDS AREA

Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e-8$ for double precision computations

speculative

- Keywords Area
- method
- conmin_mfd
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- conmin_mfd
- max_function_evaluations

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

*Alias:* none

*Argument(s):* INTEGER

*Default:* 1000

**Description**

The maximum number of function evaluations (default: 1000). See also `max.iterations`. 
scaling
  • Keywords Area
  • method
  • conmin_mfd
  • scaling

Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:
  • method_independent_controls

Specification
Alias: none
  Argument(s): none
  Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the `method`, `variables`, and `responses` blocks. When the `scaling` keyword is omitted, all `scale_types` and `scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

  • a `scales` keyword, which gives characteristic values
  • a `scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. **none** (default) - no scaling, value of `scales` keyword is ignored
2. **value** - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling

First, any characteristic values from the optional \texttt{*scales} specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

\textbf{Scales}

The \texttt{*scales} keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding \texttt{*scale\_type}, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- \texttt{none, auto, log} - optional
- \texttt{value} - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \texttt{1.0e10*DBL\_MIN}. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

\textbf{Examples}

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
linear_inequality_constraint_matrix

- Keywords Area
- method
- conmin_mfd
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

Description
In the inequality case, the constraint matrix \( A \) provides coefficients for the variables in the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_inequality_upper_bounds. The bounds, if not specified, will default to -\( \infty \), and 0, respectively, resulting in one-sided inequalities of the form

\[
Ax \leq 0.0
\]

linear_inequality_lower_bounds

- Keywords Area
- method
- conmin_mfd
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics
This keyword is related to the topics:
- linear_constraints
### Specification

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** vector values = -infinity

### Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize ($1.e+30$, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-DBL\_MAX < -\text{bigRealBoundSize}$).

### linear_inequality_upper_bounds

- **Keywords Area**  
- **method**  
- **conmin_mfd**  
- **linear_inequality_upper_bounds**

Define upper bounds for the linear inequality constraint

### Topics

This keyword is related to the topics:

- **linear_constraints**

### Specification

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** vector values = 0.
CHAPTER 6. KEYWORDS AREA

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

- Keywords Area
- method
- conmin_mfd
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_inequality_scales must be specified
6.2. METHOD

- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}_j = \frac{x_j - x_O}{x_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_inequality_scales**

- **Keywords** Area
- **method**
- **conmin_mfd**
- **linear_inequality_scales**

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

- **linear_constraints**

**Specification**

**Alias:** none
**Argument(s):** REALLIST
**Default:** vector values = 1. (no scaling)

**Description**

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`
- `'none'` - ignored
• ‘value’ - required
• ‘auto’ - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A_i} \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

**linear_equation_constraint_matrix**

- Keywords Area
- method
- conmin_mfd
- linear_equation_constraint_matrix

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$
6.2. METHOD

**linear_equation_targets**
- Keywords Area
- method
- conmin_mfd
- linear_equation_targets

Define target values for the linear equality constraints

**Topics**
This keyword is related to the topics:
- linear_constraints

**Specification**
Alias: none
- Argument(s): REALLIST
- Default: vector values = 0.

**Description**
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0.0$$

**linear_equation_scale_types**
- Keywords Area
- method
- conmin_mfd
- linear_equation_scale_types

Specify how each linear equality constraint is scaled

**Topics**
This keyword is related to the topics:
- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

linear.equality_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ‘none’ - no scaling
- ‘value’ - characteristic value if this is chosen, then linear.equality.scales must be specified
- ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

linear.equality_scales

- Keywords Area
- method
- conmin.mfd
- linear.equality.scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear.constraints
6.2. METHOD

Specification

Alias: none

Argument(s): REALIST

Default: vector values = 1. (no scaling)

Description

Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

model_pointer

- Keywords Area
- method
- conmin_mfd
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYWORDS AREA

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

evironment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
6.2. METHOD

response_functions = 3
no_gradients
no_hessians

6.2.18 conmin

- Keywords Area
- method
- conmin

Access to methods in the CONMIN library

Topics

This keyword is related to the topics:

- package_conmin

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required(Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>frcg</td>
<td>A conjugate gradient optimization method</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>mfd</td>
<td>Method of feasible directions</td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_-tolerance</td>
<td></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Compute speculative gradients</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_constraint_matrix</td>
<td>Define coefficients of the linear equalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_targets</td>
<td>Define target values for the linear equality constraints</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>linear_equation_-scale_types</th>
<th>Specify how each linear equality constraint is scaled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>linear_equation_-scales</td>
<td>Define the characteristic values to scale linear equalities.</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method.</td>
</tr>
</tbody>
</table>

**Description**

The CONMIN library[83] is a public domain library of nonlinear programming optimizers, specifically the Fletcher-Reeves conjugate gradient (Dakota’s conmin_frcg method) method for unconstrained optimization, and the method of feasible directions (Dakota’s conmin_mfd method) for constrained optimization. As CONMIN was a predecessor to the DOT commercial library, the algorithm controls are very similar.

One of the two available methods in Group 1 must be specified.

All these methods take the same Optional Keywords, dealing with linear equality and inequality constraints.

**See Also**

These keywords may also be of interest:

- dot

**frcg**

- Keywords Area
- method
- conmin
- frcg

A conjugate gradient optimization method

**Topics**

This keyword is related to the topics:

- package_conmin

**Specification**

**Alias:** none

**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description
The interpretations of the method independent controls for CONMIN are essentially identical to those for DOT.
See package dot for information related to CONMIN methods.

See Also
These keywords may also be of interest:

• mfd
• frcg

mfd

• Keywords Area
• method
• conmin
• mfd

Method of feasible directions

Topics
This keyword is related to the topics:

• package_conmin

Specification
Alias: none
Argument(s): none

Description
The interpretations of the method independent controls for CONMIN are essentially identical to those for DOT.
See package dot for information related to CONMIN methods.

See Also
These keywords may also be of interest:

• frcg
• mmfd
6.2. METHOD

max_iterations

- Keywords Area
- method
- conmin
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- conmin
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none
Argument(s): REAL
Default: 1.e-4
Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

constraint_tolerance

- Keywords Area
- method
- conmin
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: Library default
**Description**

The `constraint_tolerance` specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

**Defaults**

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e-8$ for double precision computations

**speculative**

- **Keywords Area**
- **method**
- **conmin**
- **speculative**

Compute speculative gradients

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no speculation

**Description**

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function.
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

max_function_evaluations

- Keywords Area
- method
- conmin
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.
6.2. METHOD

Scaling

- **Keywords Area**
- method
- conmin
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the `scaling` keyword is omitted, all `scale_types` and `scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `scales` keyword, which gives characteristic values
- a `scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. **none** (default) - no scaling, value of `scales` keyword is ignored
2. **value** - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{-10}$ DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
6.2. **METHOD**

**linear_inequality_constraint_matrix**

- **Keywords Area**
- **method**
- **conmin**
- **linear_inequality_constraint_matrix**

Define coefficients of the linear inequality constraints

**Topics**

This keyword is related to the topics:

- **linear_constraints**

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** no linear inequality constraints

**Description**

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by **linear_inequality_lower_bounds**, and **linear_inequality_upper_bounds**. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

**linear_inequality_lower_bounds**

- **Keywords Area**
- **method**
- **conmin**
- **linear_inequality_lower_bounds**

Define lower bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- **linear_constraints**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): REALLIST
  Default: vector values = -infinity

Description

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see *objective functions*), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \(+\text{bigRealBoundSize} \ (1.e+30, \text{as defined in Minimizer})\) are treated as \(+\infty\) and any lower bound values less than \(-\text{bigRealBoundSize}\) are treated as \(-\infty\).

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\text{DBL}_\text{MAX} < -\text{bigRealBoundSize}\)).

linear_inequality_upper_bounds

- Keywords Area
- method
- conmin
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none
  Argument(s): REALLIST
  Default: vector values = 0.
6.2. METHOD

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ (i.e. $1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

- Keywords Area
- method
- conmin
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ‘none’ - no scaling
- ‘value’ - characteristic value if this is chosen, then linear_inequality_scales must be specified
• ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^i = \frac{x^i - x^i_O}{x^i_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_inequality_scales**

- Keywords Area
- method
- conmin
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

**Description**

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
- ‘none’ - ignored
6.2. METHOD

- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_j = \frac{x_j - x_{jO}}{x_{jM}}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_{iO}) \leq a_U
\]

\[
a_L - A_i x_{iO} \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_{iO}
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

linear_equality_constraint_matrix

- Keywords Area
- method
- conmin
- linear_equality_constraint_matrix

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints

Description

In the equality case, the constraint matrix \( A \) provides coefficients for the variables on the left hand side of:

\[ A x = a_t \]
linear_equality_targets

- Keywords Area
- method
- conmin
- linear_equality_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equality_scale_types

- Keywords Area
- method
- conmin
- linear_equality_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints
6.2. METHOD

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = “none”

Description

linear.equality_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ’none’ - no scaling
- ’value’ - characteristic value if this is chosen, then linear.equality.scales must be specified
- ’auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables x:

\[ \tilde{x}^j = \frac{x^j - x^j_O}{x^j_M} \]

we have the following system for linear equality constraints

\[ a_L \leq A_i x \leq a_U \]

\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]

\[ a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \]

\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear.equality.scales

- Keywords Area
- method
- commin
- linear.equality.scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

Description

Each entry in `linear.equality.scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M - x^j_O}
$$

we have the following system for linear inequality constraints

$$
a_L \leq A_i x \leq a_U
$$

$$
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
$$

$$
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
$$

$$
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

model_pointer

- Keywords Area
- method
- conmin
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

evironment
    tabular_graphics_data
    method_pointer = 'UQ'

method
    id_method = 'UQ'
    model_pointer = 'SRR'
    sampling,
        samples = 10
        seed = 98765 rng rnum2
        response_levels = 0.1 0.2 0.6
        0.1 0.2 0.6
        0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
    id_model = 'SRR'
    surrogate global,
        dace_method_pointer = 'DACE'
        polynomial quadratic

method
    id_method = 'DACE'
    model_pointer = 'DACE_M'
    sampling sample_type lhs
        samples = 121 seed = 5034 rng rnum2

model
    id_model = 'DACE_M'
    single
    interface_pointer = 'I1'

variables
    uniform_uncertain = 2
        lower_bounds = 0. 0.
        upper_bounds = 1. 1.
        descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
6.2.19 dl_solver

- Keywords Area
- method
- dl_solver

(Experimental) Dynamically-loaded solver

Topics

This keyword is related to the topics:

- optimization_and_calibration

Specification

Alias: none

Argument(s): STRING

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>linear_inequality_-constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>linear_inequality_-lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>linear_inequality_-upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>linear_inequality_-scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>linear_inequality_-scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
</tr>
</tbody>
</table>
6.2. METHOD

### Description

This keyword specifies a dynamically-loaded optimization solver library, an experimental Dakota feature that is not enabled by default.

**linear_inequality_constraint_matrix**

- **Keywords Area**
  - method
  - dl_solver
  - linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

### Topics

This keyword is related to the topics:

- **linear_constraints**
**Specification**

Alias: none

**Argument(s):** REALLIST

**Default:** no linear inequality constraints

**Description**

In the inequality case, the constraint matrix \( A \) provides coefficients for the variables in the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where the bounds are optionally specified by `linear.inequality.lower.bounds`, and `linear.inequality.upper.bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

\[
Ax \leq 0.0
\]

**linear.inequality.lower.bounds**

- **Keywords Area**
- **method**
- **dl_solver**
- **linear.inequality.lower.bounds**

Define lower bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- **linear.constraints**

**Specification**

Alias: none

**Argument(s):** REALLIST

**Default:** vector values = -infinity

**Description**

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see `objective.functions`), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]
result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).

**linear_inequality_upper_bounds**

- Keywords Area
- method
- dl_solver
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = 0.

**Description**

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).
linear_inequality_scale_types

- Keywords Area
- method
- dl_solver
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = ”none”

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_inequality_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables x:

$$ \tilde{x}^j = \frac{x^j - x^j_O}{x^j_M} $$

we have the following system for linear inequality constraints

$$ a_L \leq A_i x \leq a_U $$
$$ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U $$
$$ a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O $$
$$ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U $$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].
6.2. METHOD

linear_inequality_scales

- Keywords: Area, method, dl_solver, linear_inequality_scales

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALIST

Default: vector values = 1. (no scaling)

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
- 'none' - ignored
- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i (\text{diag}(x_M)\tilde{x}) \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).
linear_equality_constraint_matrix

- Keywords Area
- method
- dl_solver

Define coefficients of the linear equalities

Topics
This keyword is related to the topics:
- linear_constraints

Specification

Alias: none
Argument(s): REALLIST
Default: no linear equality constraints

Description
In the equality case, the constraint matrix \( A \) provides coefficients for the variables on the left hand side of:

\[
Ax = a_t
\]

linear_equality_targets

- Keywords Area
- method
- dl_solver

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:
- linear_constraints

Specification

Alias: none
Argument(s): REALLIST
Default: vector values = 0.
Description

In the equality case, the targets \( a_t \) provide the equality constraint right hand sides:

\[
Ax = a_t
\]

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: \( Ax = 0.0 \)

linear_equation_scale_types

- Keywords Area
- method
- dl_solver
- linear_equation_scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear_equation_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_equation_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_D}{x^j_M - x^j_M}
\]

we have the following system for linear equality constraints

\[
a_L \leq A_i x \leq a_U
\]
\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_equality_scales**
- Keywords Area
- method
- dl_solver
- linear_equality_scales

Define the characteristic values to scale linear equalities

**Topics**
This keyword is related to the topics:
- linear_constraints

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** vector values = 1. (no scaling)

**Description**
Each entry in `linear_equality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.
See the scaling keyword in the `method` section for details on how to use this keyword.
Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables \( x \):
\[ \tilde{x}_j = \frac{x_j - x_{jO}}{x_{jM}} \]
we have the following system for linear inequality constraints
\[ a_L \leq A_i x \leq a_U \]
\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]
and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].
6.2. METHOD

max_function_evaluations

- Keywords Area
- method
- dl_solver
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling

- Keywords Area
- method
- dl_solver
- scaling

Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling
CHAPTER 6. KEYWORDS AREA

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the `scaling` keyword is omitted, all `_scale_types` and `*_scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `*_scales` keyword, which gives characteristic values
- a `*_scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. `none` (default) - no scaling, value of `*_scales` keyword is ignored
2. `value` - multiplicative scaling
3. `auto` - automatic scaling
4. `log` - logarithmic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*_scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*_scale_type`, as described above.

Depending on the scale type, the characteristic values may be required or optional.
6.2. **METHOD**

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0 \times 10^{10} \times DBL_{MIN}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

**model_pointer**

- **Keywords Area**
- **method**
- **dl_solver**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The **model_pointer** is used to specify which **model** block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a **model** block in the Dakota input file that has a corresponding **id_model** with the same name.
**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
                  0.1 0.2 0.6
                  0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians
```

---

6.2.20 **npsol_sqp**

- Keywords Area
6.2. METHOD

- method
- npsol_sqp

Sequential Quadratic Program

Topics

This keyword is related to the topics:

- package_npsol
- sequential_quadratic_programming
- local_optimization_methods

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>verify_level</td>
<td>Verify the quality of analytic gradients</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>function_precision</td>
<td>Specify the maximum precision of the analysis code responses</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>linesearch_-tolerance</td>
<td>Choose how accurately the algorithm will compute the minimum in a line search</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
<td>---------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Compute speculative gradients</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_constraint_matrix</td>
<td>Define coefficients of the linear equalities</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

| Optional | linear_equality_-targets | Define target values for the linear equality constraints
| Optional | linear_equality_-scale_types | Specify how each linear equality constraint is scaled
| Optional | linear_equality_-scales | Define the characteristic values to scale linear equalities
| Optional | model_pointer | Identifier for model block to be used by a method

Description

NPSOL provides an implementation of sequential quadratic programming that can be accessed with npsol_sqp.

Stopping Criteria

The method independent controls for max_iterations and max_function_evaluations limit the number of major SQP iterations and the number of function evaluations that can be performed during an NPSOL optimization. The convergence_tolerance control defines NPSOL’s internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function). The constraint_tolerance control defines how tightly the constraint functions are satisfied at convergence. The default value is dependent upon the machine precision of the platform in use, but is typically on the order of 1.e-8 for double precision computations. Extremely small values for constraint_tolerance may not be attainable. The output verbosity setting controls the amount of information generated at each major SQP iteration: the silent and quiet settings result in only one line of diagnostic output for each major iteration and print the final optimization solution, whereas the verbose and debug settings add additional information on the objective function, constraints, and variables at each major iteration.

Concurrency

NPSOL is not a parallel algorithm and cannot directly take advantage of concurrent evaluations. However, if numerical_gradients with method_source dakota is specified, then the finite difference function evaluations can be performed concurrently (using any of the parallel modes described in the Users Manual[4]).

An important related observation is the fact that NPSOL uses two different line searches depending on how gradients are computed. For either analytic_gradients or numerical_gradients with method_source dakota, NPSOL is placed in user-supplied gradient mode (NPSOL’s “Derivative Level” is set to 3) and it uses a gradient-based line search (the assumption is that user-supplied gradients are inexpensive). On the other hand, if numerical_gradients are selected with method_source vendor, then NPSOL is computing finite differences internally and it will use a value-based line search (the assumption is that finite differencing on each line search evaluation is too expensive). The ramifications of this are: (1) performance will vary between method_source dakota and method_source vendor for numerical_gradients, and (2) gradient speculation is unnecessary when performing optimization in parallel since the gradient-based line search in user-supplied gradient mode is already load balanced for parallel execution. Therefore, a speculative specification will be ignored by NPSOL, and optimization with numerical gradients should select method_source dakota for load balanced parallel operation and method_source vendor for efficient serial operation.
Linear constraints
Lastly, NPSOL supports specialized handling of linear inequality and equality constraints. By specifying the coefficients and bounds of the linear inequality constraints and the coefficients and targets of the linear equality constraints, this information can be provided to NPSOL at initialization and tracked internally, removing the need for the user to provide the values of the linear constraints on every function evaluation.

verify_level

- Keywords Area
- method
- npsol_sqp
- verify_level

Verify the quality of analytic gradients

Specification
Alias: none
Argument(s): INTEGER
Default: -1 (no gradient verification)

Description
verify_level instructs the NPSOL and NLSSOL algorithms to perform their own finite difference verification of the gradients provided by Dakota. Typically these are used to verify analytic_gradients produced by a simulation code, though the option can be used with other Dakota-supplied gradient types including numerical or mixed.
Level 1 will verify the objective gradients, level 2, the nonlinear constraint gradients, and level 3, both. See the Optional Input Parameters section of the NPSOL manual[33] for additional information, including options to verify at the user-supplied initial point vs. first feasible point.

function_precision

- Keywords Area
- method
- npsol_sqp
- function_precision

Specify the maximum precision of the analysis code responses

Specification
Alias: none
Argument(s): REAL
Default: 1.0e-10
6.2. METHOD

Description

The function_precision control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.

linesearch_tolerance

- Keywords Area
- method
- npsol_sqp
- linesearch_tolerance

Choose how accurately the algorithm will compute the minimum in a line search

Specification

Alias: none

Argument(s): REAL

Default: 0.9 (inaccurate line search)

Description

The linesearch_tolerance setting controls the accuracy of the line search. The smaller the value (between 0 and 1), the more accurately the algorithm will attempt to compute a precise minimum along the search direction.

convergence_tolerance

- Keywords Area
- method
- npsol_sqp
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4
**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a *relative convergence tolerance* for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

**max_iterations**

- Keywords Area
- method
- npsol_sqp
- `max_iterations`

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global:{reliability, interval_est, evidence} / efficient_global: 25*n)
6.2. METHOD

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

constraint_tolerance
- Keywords Area
- method
- npsol_sqp
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: Library default

Description
The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults
Defaults can vary depending on the method.
- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e-8$ for double precision computations

speculative
- Keywords Area
- method
- npsol_sqp
- speculative

Compute speculative gradients
This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

- Argument(s): none
- Default: no speculation

**Description**

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- npsol_sqp
- max_function_evaluations
6.2. METHOD

Stopping criteria based on number of function evaluations

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none  
Argument(s): INTEGER  
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling
- Keywords Area
- method
- npsol_sqp
- scaling

Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none  
Argument(s): none  
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.
Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling
   
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   
   - two-sided bounds scaled into the interval [0,1];
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling
   
   First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.
   
   Logarithmic scaling is not available for linear constraints.
   
   When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
6.2. METHOD

Examples
The two examples below are equivalent:

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100

linear_inequality_constraint_matrix

• Keywords Area
• method
• npsol_sqp
• linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics
This keyword is related to the topics:

• linear_constraints

Specification

Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

Description
In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$ a_l \leq Ax \leq a_u $$

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_inequality_upper_bounds. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$ Ax \leq 0.0 $$

.
linear_inequality_lower_bounds

- Keywords Area
- method
- npsol_sqp
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** vector values = -infinity

**Description**

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

\[ a_l \leq Ax \leq a_u \]

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[ Ax \leq 0.0 \]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- npsol_sqp
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint
6.2. METHOD

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ (1.e+30, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types
- Keywords Area
- method
- npsol_sqp
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"
Description

`linear_inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear_inequality_scales` must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied *after* any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - \bar{x}^j}{\bar{x}_M^j}
$$

we have the following system for linear inequality constraints

$$
a_L \leq A_i x \leq a_U
$$

$$
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + \bar{x}_O) \leq a_U
$$

$$
a_L - A_i \bar{x}_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i \bar{x}_O
$$

$$
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

`linear_inequality_scales`

- **Keywords Area**
- **method**
- **npsol, sqp**
- **linear_inequality_scales**

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- **linear_constraints**

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)
6.2. METHOD

Description

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality.scale_type`:

- scale_type - behavior of `linear_inequality_scales`
- 'none' - ignored
- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - x_J^j}{x_M^j}
$$

we have the following system for linear inequality constraints

$$
a_L \leq A_i x \leq a_U \\
a_L \leq A_i \left( \text{diag}(x_M)^{\tilde{x}} + x_O \right) \leq a_U \\
a_L - A_i x_O \leq A_i \text{diag}(x_M)^{\tilde{x}} \leq a_U - A_i x_O \\
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_equality_constraint_matrix

- Keywords Area
- method
- npsol_sqp
- linear_equality_constraint_matrix

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints
CHAPTER 6. KEYWORDS AREA

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

linear.equality_targets

- Keywords Area
- method
- npsol_sqp
- linear.equality_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none

Argument(s): REALLIST
Default: vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear.equality.scale_types

- Keywords Area
- method
- npsol_sqp
- linear.equality.scale_types

Specify how each linear equality constraint is scaled
6.2. METHOD

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear.equality_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear.equality.scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear.equality_scales

- Keywords Area
- method
- npsol_sqp
- linear.equality_scales

Define the characteristic values to scale linear equalities
Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 1 . (no scaling)

Description
Each entry in linearequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.
See the scaling keyword in the method section for details on how to use this keyword.
Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i\tilde{x} \leq a_U$$
$$a_L \leq A_i(diag(x_M)\tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i(diag(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i\tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

model_pointer

- Keywords Area
- method
- npsol_sqp
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer
6.2. METHOD

Specification

Alias: none
  Argument(s): STRING
  Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations
needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model
block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-
_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'
```
interface
    id_interface = 'I1'
    system asynch evalution_concurrency = 5
    analysis_driver = 'text_book'

responses
    response_functions = 3
    no_gradients
    no_hessians

6.2.21 nlssol_sqp

- Keywords Area
- method
- nlsol_sqp

Sequential Quadratic Program for nonlinear least squares

Topics
This keyword is related to the topics:

- sequential_quadratic_programming
- nonlinear_least_squares

Specification
Alias: none
Argument(s): none

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
|----------- | Group         |               | Description   |
| Optional  | verify_level  |               | Verify the quality of analytic gradients |
| Optional  | function__precision |               | Specify the maximum precision of the analysis code responses |
| Optional  | linesearch__tolerance |               | Choose how accurately the algorithm will compute the minimum in a line search |
### Optional 6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>convergence_-tolerance</th>
<th>Stopping criterion based on convergence of the objective function or statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_-tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_-evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>linear.equality_-constraint_matrix</th>
<th>Define coefficients of the linear equalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>linear.equality_-targets</td>
<td>Define target values for the linear equality constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>linear.equality_-scale_types</td>
<td>Specify how each linear equality constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td>linear.equality_-scales</td>
<td>Define the characteristic values to scale linear equalities</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

NLSSOL is available as `nlssol_sqp` and supports unconstrained, bound-constrained, and generally-constrained problems. It exploits the structure of a least squares objective function through the periodic use of Gauss-Newton Hessian approximations to accelerate the SQP algorithm.

**Stopping Criteria**

The method independent controls for `max_iterations` and `max_function_evaluations` limit the number of major SQP iterations and the number of function evaluations that can be performed during an NPSOL optimization. The `convergence_tolerance` control defines NPSOL’s internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function). The `constraint_tolerance` control defines how tightly the constraint functions are satisfied at convergence. The default value is dependent upon the machine precision of the platform in use, but is typically on the order of `1.e-8` for double precision computations. Extremely small values for `constraint_tolerance` may not be attainable.

**See Also**

These keywords may also be of interest:

- `npsol_sqp`
- `nl2sol`
- `optpp_g_newton`
- `field_calibration_terms`

**verify_level**

- `Keywords Area`
6.2. METHOD

- method
- nlssol_sqp
- verify_level

Verify the quality of analytic gradients

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** -1 (no gradient verification)

**Description**

`verify_level` instructs the NPSOL and NLSSOL algorithms to perform their own finite difference verification of the gradients provided by Dakota. Typically these are used to verify analytic gradients produced by a simulation code, though the option can be used with other Dakota-supplied gradient types including numerical or mixed.

Level 1 will verify the objective gradients, level 2, the nonlinear constraint gradients, and level 3, both. See the Optional Input Parameters section of the NPSOL manual[33] for additional information, including options to verify at the user-supplied initial point vs. first feasible point.

**function_precision**

- Keywords Area
- method
- nlssol_sqp
- function_precision

Specify the maximum precision of the analysis code responses

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** 1.0e-10

**Description**

The `function_precision` control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.
linesearch_tolerance

- Keywords Area
- method
- nlssol_sqp
- linesearch_tolerance

Choose how accurately the algorithm will compute the minimum in a line search

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 0.9 (inaccurate line search)

**Description**

The `linesearch_tolerance` setting controls the accuracy of the line search. The smaller the value (between 0 and 1), the more accurately the algorithm will attempt to compute a precise minimum along the search direction.

convergence_tolerance

- Keywords Area
- method
- nlssol_sqp
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 1.e-4
6.2. **METHOD**

**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

**max_iterations**

- **Keywords Area**
- **method**
- **nlssol_sqp**
- **max_iterations**

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)
CHAPTER 6. KEYWORDS AREA

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

constraint_tolerance

- Keywords Area
- method
- nlssol_sqp
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics

This keyword is related to the topics:

- method/independent_controls

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of 1.e-8 for double precision computations

speculative

- Keywords Area
- method
- nlssol_sqp
- speculative

Compute speculative gradients
6.2. METHOD

Topics
This keyword is related to the topics:
- method
- independent
- controls

Specification
Alias: none
Argument(s): none
Default: no speculation

Description
When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose.

In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

max_function_evaluations
- Keywords Area
- method
- nlssol_sqp
- max_function_evaluations
Stopping criteria based on number of function evaluations

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling
- Keywords Area
- method
- nlssol_sqp
- scaling

Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.
Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all scale_types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
Examples
The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

**linear_inequality_constraint_matrix**

- Keywords Area
- method
- nlssol_sqp
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

**Argument(s):** REALLIST

**Default:** no linear inequality constraints

Description

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$ a_l \leq Ax \leq a_u $$

Where the bounds are optionally specified by `linear_inequality_lower_bounds`, and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$ Ax \leq 0.0 $$
6.2. METHOD

linear_inequality_lower_bounds

- Keywords Area
- method
- nlssol_sqp
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL MAX} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- nlssol_sqp
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint
Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\text{infinity}$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\text{infinity}$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

- Keywords Area
- method
- nlssol_sqp
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = ”none”
6.2. METHOD

Description

`linear_inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear_inequality_scales` must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \(x\):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\operatorname{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \operatorname{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

`linear_inequality_scales`

- **Keywords** Area
- **method**
- **nlssol_sqp**
- **linear_inequality_scales**

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- **linear_constraints**

Specification

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = 1 . (no scaling)
Description

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`
- `'none'` - ignored
- `'value'` - required
- `'auto'` - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
$$

we have the following system for linear inequality constraints

$$
a_L \leq A_i x \leq a_U
$$

$$
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
$$

$$
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
$$

$$
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

**linear_equality_constraint_matrix**

- Keywords Area
- method
- `nlssol_sqp`
- `linear_equality_constraint_matrix`

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- `linear_constraints`

Specification

Alias: none

Argument(s): `REALLIST`

Default: no linear equality constraints
6.2. METHOD

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

linear_equalities_targets

- Keywords Area
- method
- nlssol_sqp
- linear_equalities_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

**Argument(s):** REALLIST

**Default:** vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equalities_scale_types

- Keywords Area
- method
- nlssol_sqp
- linear_equalities_scale_types

Specify how each linear equality constraint is scaled
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description
linear_equalities_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.
An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_equalities_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}_j = \frac{x_j - x_O}{x_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_equalities_scales

- Keywords Area
- method
- nlssol_sqp
- linear_equalities_scales

Define the characteristic values to scale linear equalities
6.2. \textit{METHOD}

\textbf{Topics}

This keyword is related to the topics:

- \texttt{linear\_constraints}

\textbf{Specification}

\textbf{Alias:} none

\textbf{Argument(s):} REALLIST

\textbf{Default:} vector values = 1. (no scaling)

\textbf{Description}

Each entry in \texttt{linear\_equality\_scales} may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the \texttt{method} section for details on how to use this keyword.

Scaling for linear constraints is applied \textit{after} any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
$$

we have the following system for linear inequality constraints

$$
a_L \leq A_i x \leq a_U
$$

$$
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
$$

$$
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

\textbf{model\_pointer}

- Keywords Area
- method
- \texttt{nlssol\_sqp}
- \texttt{model\_pointer}

Identifier for model block to be used by a method

\textbf{Topics}

This keyword is related to the topics:

- \texttt{block\_pointer}
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
    sample_type lhs
distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'
interface
    id_interface = 'I1'
    system async evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
    response_functions = 3
    no_gradients
    no_hessians

6.2.22 stanford

- Keywords Area
- method
- stanford

Select methods from the Stanford package

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required(Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npsol</td>
<td>Duplicate of method-npsol-sqp</td>
<td>npsol</td>
<td>Duplicate of method-nlssol-sqp</td>
</tr>
<tr>
<td>optional</td>
<td>verify_level</td>
<td>Optional</td>
<td>Verify the quality of analytic gradients</td>
</tr>
<tr>
<td>optional</td>
<td>function_precision</td>
<td>Optional</td>
<td>Specify the maximum precision of the analysis code responses</td>
</tr>
<tr>
<td>optional</td>
<td>linesearch_-tolerance</td>
<td>Optional</td>
<td>Choose how accurately the algorithm will compute the minimum in a line search</td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_tolerance</td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Compute speculative gradients</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

Optional

linear_equality_-constraint_matrix
Define coefficients of the linear equalities

Optional

linear_equality_-targets
Define target values for the linear equality constraints

Optional

linear_equality_-scale_types
Specify how each linear equality constraint is scaled

Optional

linear_equality_-scales
Define the characteristic values to scale linear equalities

Optional

model_pointer
Identifier for model block to be used by a method

Description

This is a duplicate. See the method pages here:

- nlsol_sqp
- npsol_sqp

npsol

- Keywords Area
- method
- stanford
- npsol

Duplicate of method-npsol_sqp

Specification

Alias: none
Argument(s): none

Description

See the page for npsol_sqp
nlssol

- Keywords Area
- method
- stanford
- nlssol

Duplicate of method-nlssol_sqp

**Specification**

Alias: none

Argument(s): none

**Description**

See the page for nlssol_sqp

verify_level

- Keywords Area
- method
- stanford
- verify_level

Verify the quality of analytic gradients

**Specification**

Alias: none

Argument(s): INTEGER

Default: -1 (no gradient verification)

**Description**

verify_level instructs the NPSOL and NLSSOL algorithms to perform their own finite difference verification of the gradients provided by Dakota. Typically these are used to verify analytic_gradients produced by a simulation code, though the option can be used with other Dakota-supplied gradient types including numerical or mixed.

Level 1 will verify the objective gradients, level 2, the nonlinear constraint gradients, and level 3, both. See the Optional Input Parameters section of the NPSOL manual[33] for additional information, including options to verify at the user-supplied initial point vs. first feasible point.
6.2. METHOD

function_precision

- Keywords Area
- method
- stanford
- function_precision

Specify the maximum precision of the analysis code responses

Specification

Alias: none
Argument(s): REAL
Default: 1.0e-10

Description

The function_precision control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.

linesearch_tolerance

- Keywords Area
- method
- stanford
- linesearch_tolerance

Choose how accurately the algorithm will compute the minimum in a line search

Specification

Alias: none
Argument(s): REAL
Default: 0.9 (inaccurate line search)

Description

The linesearch_tolerance setting controls the accuracy of the line search. The smaller the value (between 0 and 1), the more accurately the algorithm will attempt to compute a precise minimum along the search direction.
convergence_tolerance

- Keywords Area
- method
- stanford
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

Argument(s): REAL

Default: 1.e-4

**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol
max_iterations

- Keywords Area
- method
- stanford
- max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

constraint_tolerance

- Keywords Area
- method
- stanford
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: Library default
CHAPTER 6. KEYWORDS AREA

Description

The `constraint_tolerance` specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of $1.e-8$ for double precision computations

speculative

- Keywords Area
- method
- stanford
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function
values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

max_function_evaluations

- **Keywords Area**
- **method**
- **stanford**
- **max_function_evaluations**

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- **method_independent_controls**

Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.
scaling

- Keywords Area
- method
- stanford
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all *scales and ∗scale types specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. **auto** - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval \([0,1]\);
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into \([0,1]\).

4. **log** - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- **none, auto, log** - optional
- **value** - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0\times10^{\text{DBL_MIN}}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```
linear_inequality_constraint_matrix

- Keywords Area
- method
- stanford
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

**Topics**
This keyword is related to the topics:
- linear_constraints

**Specification**
Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

**Description**
In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by **linear_inequality_lower_bounds**, and **linear_inequality_upper_bounds**. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

linear_inequality_lower_bounds

- Keywords Area
- method
- stanford
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

**Topics**
This keyword is related to the topics:
- linear_constraints
6.2. Method

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.0e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- stanford
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.
Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see \texttt{objective.functions}), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ (1.e+30, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

\texttt{linear.inequality.scale_types}

- \texttt{Keywords Area}
- \texttt{method}
- \texttt{stanford}
- \texttt{linear.inequality.scale_types}

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- \texttt{linear.constraints}

Specification

\textbf{Alias}: none

\textbf{Argument(s)}: STRINGLIST

\textbf{Default}: vector values = ”none”

Description

\texttt{linear.inequality.scale_types} provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ’none’ - no scaling
- ’value’ - characteristic value if this is chosen, then \texttt{linear.inequality.scales} must be specified
6.2. METHOD

• ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_inequality_scales**

- Keywords Area
- method
- stanford
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

Argument(s): REALIST

Default: vector values = 1. (no scaling)

**Description**

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`

- ‘none’ - ignored
• 'value' - required
• 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.
Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_\text{O}}{x^j_\text{M}}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_\text{M}) \tilde{x} + x_\text{O}) \leq a_U$$
$$a_L - A_i x_\text{O} \leq A_i \text{diag}(x_\text{M}) \tilde{x} \leq a_U - A_i x_\text{O}$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_equality_constraint_matrix**

- **Keywords Area**
- **method**
- **stanford**
- **linear_equality_constraint_matrix**

Define coefficients of the linear equalities

**Topics**

This keyword is related to the topics:

- **linear_constraints**

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** no linear equality constraints

**Description**

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$
6.2. METHOD

linear_equali

ty_targets

- Keywords Area
- method
- stanford
- linear_equali

ty_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST
Default: vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equali

ty_scale_types

- Keywords Area
- method
- stanford
- linear_equali

ty_scale_types

Specify how each linear equality constraint is scaled

Topics
This keyword is related to the topics:

- linear_constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

linear.equality.scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ‘none’ - no scaling
- ‘value’ - characteristic value if this is chosen, then linear.equality.scales must be specified
- ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[ \tilde{x}^j = \frac{x^j - x^j_O}{x^j_M} \]

we have the following system for linear equality constraints

\[ a_L \leq A_i x \leq a_U \]
\[ a_L \leq A_i \left( \text{diag}(x_M) \tilde{x} + x_O \right) \leq a_U \]
\[ a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \]
\[ \hat{a}_L \leq \hat{A}_i \tilde{x} \leq \hat{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

linear.equality.scales

- Keywords Area
- method
- stanford
- linear.equality.scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear.constraints
6.2. METHOD

Specification

Alias: none
Argument(s): REALIST
Default: vector values = 1. (no scaling)

Description

Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint. See the scaling keyword in the method section for details on how to use this keyword. Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x_O}{x_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

model_pointer

- Keywords Area
- method
- stanford
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYWORDS AREA

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

```
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SRRR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SRRR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.23 nlpql.sqp

- Keywords Area
- method
- nlpql.sqp

Sequential Quadratic Program

Topics

This keyword is related to the topics:

- package.nlpl
- sequential.quadratic.programming
- local.optimization.methods

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>linear_inequality_-constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
</tr>
</tbody>
</table>
### Description

NLPQL provides an implementation of sequential quadratic programming through `nlpqpsqp`. The particular SQP implementation in `nlpq一期` uses a variant with distributed and non-monotone line search. Thus, this variant is designed to be more robust in the presence of inaccurate or noisy gradients common in many engineering applications.

The method independent controls for maximum iterations and output verbosity are mapped to NLPQL controls `MAXIT` and `IPRINT`, respectively. The maximum number of function evaluations is enforced within the NLPQ-
6.2. METHOD

LPOptimizer class.

**max_iterations**
- Keywords Area
- method
- nlpql_sqp
- max_iterations

Stopping criterion based on number of iterations

**Topics**
This keyword is related to the topics:
- method independent controls

**Specification**
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)

**Description**
The maximum number of iterations (default: 100). See also max_function_evaluations.

**convergence_tolerance**
- Keywords Area
- method
- nlpql_sqp
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**
This keyword is related to the topics:
- method independent controls

**Specification**
Alias: none
Argument(s): REAL
Default: 1.e-4
CHAPTER 6. KEYWORDS AREA

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximates the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See `nl2sol`

linear_inequality_constraint_matrix

- Keywords Area
- method
- nlpql_sqp
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear inequality constraints
Description

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by \texttt{linear.inequality.lower.bounds}, and \texttt{linear.inequality.upper.bounds}. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

\texttt{linear.inequality.lower.bounds}

- Keywords Area
- method
- \texttt{nlpql.sqp}
- \texttt{linear.inequality.lower.bounds}

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

\texttt{Alias:} none

\texttt{Argument(s):} \texttt{REALLIST}

\texttt{Default:} vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see \texttt{objective.functions}), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +\texttt{bigRealBoundSize} (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -\texttt{bigRealBoundSize} are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \texttt{-DBL_MAX} < -\texttt{bigRealBoundSize}).
linear_inequality_upper_bounds

- Keywords Area
- method
- nlpql_sqp
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

- Keywords Area
- method
- nlpql_sqp
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description
linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_inequality_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M - x^j_O}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i (\text{diag}(x_M)\tilde{x} - A_i x_O)
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

linear_inequality_scales

- Keywords Area
- method
- nlpql_sqp
- linear_inequality_scales

Define the characteristic values to scale linear inequalities
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none

Argument(s): REALIST

Default: vector values = 1 . (no scaling)

Description
Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
- ‘none’ - ignored
- ‘value’ - required
- ‘auto’ - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M - x^j_O}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U \\
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \\
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_equation_constraint_matrix

- Keywords Area
- method
- nlpql_sqp
- linear_equation_constraint_matrix

Define coefficients of the linear equalities
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
- Argument(s): REALLIST
- Default: no linear equality constraints

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

linear_equality_targets
- Keywords Area
- method
- nlpql_sqp
- linear_equality_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
- Argument(s): REALLIST
- Default: vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0.0$$
linear_equation_scale_types

- Keywords Area
- method
- nlpql_sqp
- linear_equation_scale_types

Specify how each linear equality constraint is scaled

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description
linear_equation_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:
- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_equation_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_{O}}{x^j_{M}} 
\]

we have the following system for linear equality constraints

\[
a_L \leq A_i x \leq a_U \\
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \\
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U 
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).
6.2. METHOD

linear_equality_scales

- Keywords Area
- method
- nlql_sqp
- linear_equality_scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)

Description

Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i (\text{diag}(x_M)\tilde{x} - A_i x_O)
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).
max_function_evaluations

- Keywords Area
- method
- nlpql_sqp
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
Argument(s): INTEGER
Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.

d\hspace{1cm}scale\hspace{1cm}ing

- Keywords Area
- method
- nlpql_sqp
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
Argument(s): none
Default: no scaling
6.2. METHOD

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the `method`, `variables`, and `responses` blocks. When the `scaling` keyword is omitted, all `scale_types` and `*scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `*scales` keyword, which gives characteristic values
- a `*scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. `none` (default) - no scaling, value of `*scales` keyword is ignored
2. `value` - multiplicative scaling
3. `auto` - automatic scaling
4. `log` - logarithmic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval \([0,1]\);
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into \([0,1]\).

4. `log` - logarithmic scaling

First, any characteristic values from the optional `*scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_type`, as described above.

Depending on the scale type, the characteristic values may be required or optional.
• none, auto, log - optional
• value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{10} \times \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**
The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

**model_pointer**

• **Keywords Area**
• **method**
• **nlpql_sqp**
• **model_pointer**

  Identifier for model block to be used by a method

**Topics**
This keyword is related to the topics:

• **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**
The **model_pointer** is used to specify which **model** block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a **model** block in the Dakota input file that has a corresponding **id_model** with the same name.
6.2. METHOD

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative. See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.24 optpp_cg

  Keywords Area
A conjugate gradient optimization method

### Topics
This keyword is related to the topics:
- `package_optpp`
- `local_optimization_methods`

### Specification
**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><code>max_step</code></td>
<td>Max change in design point</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>gradient_tolerance</code></td>
<td>Stopping criteria based on L2 norm of gradient</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>max_iterations</code></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>convergence_tolerance</code></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>speculative</code></td>
<td>Compute speculative gradients</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>max_function_evaluations</code></td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
</tbody>
</table>
### Description

The conjugate gradient method is an implementation of the Polak-Ribiere approach and handles only unconstrained problems.

See [package_optpp](#) for info related to all optpp methods.
See Also

These keywords may also be of interest:

- optpp.g.newton
- optpp.pds
- optpp_fd.newton
- optpp.newton
- optpp.g.newton

max_step

- Keywords Area
- method
- optpp.cg
- max_step

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The max_step control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The gradient_tolerance control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The gradient_tolerance control is defined for all gradient-based optimizers.

gradient_tolerance

- Keywords Area
- method
- optpp.cg
- gradient_tolerance

Stopping criteria based on L2 norm of gradient
6.2. METHOD

Specification

Alias: none
Argument(s): REAL
Default: \(1.e^{-4}\)

Description

The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The `gradient_tolerance` control is defined for all gradient-based optimizers.

`max_iterations`

- **Keywords Area**
- **method**
- **optpp_cg**
- **max_iterations**

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- **method_independent_controls**

Specification

Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: `fsu_cvt`, local_reliability: 25; global{reliability, interval_est, evidence} / efficient-global: 25\(\ast n\))

Description

The maximum number of iterations (default: 100). See also `max_function_evaluations`.

`convergence_tolerance`

- **Keywords Area**
- **method**
- **optpp_cg**
- **convergence_tolerance**

Stopping criterion based on convergence of the objective function or statistics
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- method.independent_controls

Specification
Alias: none
Argument(s): REAL
Default: 1.e-4

Description
The convergence.tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence.tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library
This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence.tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence.tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

speculative

- Keywords Area
- method
- optpp_cg
- speculative

Compute speculative gradients
6.2. METHOD

Topics
This keyword is related to the topics:

- method
- independent
- controls

Specification
Alias: none
Argument(s): none
Default: no speculation

Description
When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose.

In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- optpp_cg
- max_function_evaluations
Stopping criteria based on number of function evaluations

**Topics**
This keyword is related to the topics:
- `method_independent_controls`

**Specification**

Alias: none  
**Argument(s):** INTEGER  
**Default:** 1000

**Description**
The maximum number of function evaluations (default: 1000). See also `max_iterations`.

**scaling**
- `Keywords Area`
- `method`
- `optpp_cg`
- `scaling`

Turn on scaling for variables, responses, and constraints

**Topics**
This keyword is related to the topics:
- `method_independent_controls`

**Specification**

Alias: none  
**Argument(s):** none  
**Default:** no scaling

**Description**
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.
6.2. METHOD

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   - two-sided bounds scaled into the interval [0,1];
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling
   First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.
   Logarithmic scaling is not available for linear constraints.
   When continuous design variables are log scaled, linear constraints are not allowed.

Scales
The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
Examples

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

linear_inequality_constraint_matrix

- **Keywords Area**
- **method**
- **optpp_cg**
- **linear_inequality_constraint_matrix**

Define coefficients of the linear inequality constraints

Topics

This keyword is related to the topics:

- **linear_constraints**

Specification

**Alias:** none

**Argument(s):** REALIST

**Default:** no linear inequality constraints

Description

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by **linear_inequality_lower_bounds**, and **linear_inequality_upper_bounds**. The bounds, if not specified, will default to -$\infty$ and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$
6.2. METHOD

linear_inequality_lower_bounds

- Keywords Area
- method
- optpp_cg
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:
- linear_constraints

Specification

Alias: none

Argument(s): REALLIST
Default: vector values = -infinity

Description

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \( +\text{bigRealBoundSize} \) (i.e., 30, as defined in Minimizer) are treated as +infinity and any lower bound values less than \( -\text{bigRealBoundSize} \) are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \( -\text{DBL_MAX} < -\text{bigRealBoundSize} \)).

linear_inequality_upper_bounds

- Keywords Area
- method
- optpp_cg
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types
- Keywords Area
- method
- optpp.cg
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"
6.2. METHOD

Description

*linear_inequality_scale_types* provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling. An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then *linear_inequality_scales* must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \(x\):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x_M^j}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

*linear_inequality_scales*

- **Keywords** Area
- **method**
- **optpp_cg**
- **linear_inequality_scales**

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- **linear_constraints**

Specification

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = 1. (no scaling)
CHAPTER 6. KEYWORDS AREA

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
  - 'none' - ignored
  - 'value' - required
  - 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

linear_equality_constraint_matrix

- Keywords Area
- method
- optpp_cg
- linear_equality_constraint_matrix

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints
6.2. METHOD

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

\[ Ax = a_t \]

\textbf{linear\_equality\_targets}

- Keywords Area
- method
- optpp\_cg
- linear\_equality\_targets

Define target values for the linear equality constraints

\textbf{Topics}

This keyword is related to the topics:

- linear\_constraints

\textbf{Specification}

\textbf{Alias:} none

\textbf{Argument(s):} REALLIST

\textbf{Default:} vector values = 0.

\textbf{Description}
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

\[ Ax = a_t \]

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

\[ Ax = 0.0 \]

\textbf{linear\_equality\_scale\_types}

- Keywords Area
- method
- optpp\_cg
- linear\_equality\_scale\_types

Specify how each linear equality constraint is scaled
Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = ”none”

Description

linear_equalities_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ’none’ - no scaling
- ’value’ - characteristic value if this is chosen, then linear_equalities_scales must be specified
- ’auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}_j = \frac{x_j - x_{O}}{x_{M}}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_{O}) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_equalities_scales

- Keywords Area
- method
- optpp_cg
- linear_equalities_scales

Define the characteristic values to scale linear equalities
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 1. (no scaling)

Description
Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.
Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables x:

\[
\tilde{x}_j = \frac{x_j - x_j^O}{x_j^M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

model_pointer

- Keywords Area
- method
- optpp.cg
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
        0.1 0.20.6
            0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'II'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'
6.2. METHOD

interface
id_interface = 'I1'
system async evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.25 optpp_q_newton

- Keywords Area
- method
- optpp_q_newton

Quasi-Newton optimization method

Topics
This keyword is related to the topics:
- package_optpp
- local_optimization_methods

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>search_method</td>
<td>search_method</td>
<td>Select a search method for Newton-based optimizers</td>
</tr>
<tr>
<td>Optional</td>
<td>merit_function</td>
<td>merit_function</td>
<td>Balance goals of reducing objective function and satisfying constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>steplength_to_boundary</td>
<td>steplength_to_boundary</td>
<td>Controls how close to the boundary of the feasible region the algorithm is allowed to move</td>
</tr>
<tr>
<td>Optional</td>
<td>centering-parameter</td>
<td>Controls how closely the algorithm should follow the &quot;central path&quot;</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_step</td>
<td>Max change in design point Stopping criteria based on L2 norm of gradient</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>gradient_tolerance</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>convergence-tolerance</td>
<td>Compute speculative gradients</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function-evaluations</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality-constraint_matrix</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality-lower_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>linear_inequality_scale_types</th>
<th>Specify how each linear inequality constraint is scaled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>linear_inequality_scales</td>
<td>Define the characteristic values to scale linear inequalities.</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_constraint_matrix</td>
<td>Define coefficients of the linear equalities.</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_targets</td>
<td>Define target values for the linear equality constraints.</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_scale_types</td>
<td>Specify how each linear equality constraint is scaled.</td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equality_scales</td>
<td>Define the characteristic values to scale linear equalities.</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method.</td>
</tr>
</tbody>
</table>

Description

This is a Newton method that expects a gradient and computes a low-rank approximation to the Hessian. Each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a nonlinear interior-point approach to manage the constraints.

See package.optpp for info related to all optpp methods.

search_method

- Keywords Area
- method
- optpp_q_newton
- search_method

Select a search method for Newton-based optimizers.

Specification

Alias: none
Argument(s): none
<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>value_based_line_search</td>
<td>Use only function values for line search</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gradient_based_line_search</td>
<td>Set the search method to use the gradient</td>
</tr>
<tr>
<td></td>
<td></td>
<td>trust_region</td>
<td>Use trust region as the globalization strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tr_pds</td>
<td>Use direct search as the local search in a trust region method</td>
</tr>
</tbody>
</table>

**Description**

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by [61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_based_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

**value_based_line_search**

- Keywords Area
- method
- optpp_q_newton
- search_method
- value_based_line_search

Use only function values for line search

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** trust_region (unconstrained), value_based_line_search (bound/general constraints)
6.2. METHOD

Description

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_based_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

gradient_based_line_search

- Keywords Area
- method
- optpp.q_newton
- search_method
- gradient_based_line_search

Set the search method to use the gradient

Specification

Alias: none
Argument(s): none

Description

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_based_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

trust_region

- Keywords Area
- method
Use trust region as the globalization strategy.

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

**tr_pds**

- `Keywords Area`
- `method`
- `optpp_q_newton`
- `search_method`
- `tr_pds`

Use direct search as the local search in a trust region method.

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by[6]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_based_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function
and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

**merit_function**

- **Keywords Area**
- **method**
- **optpp_q_newton**
- **merit_function**

Balance goals of reducing objective function and satisfying constraints

### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** `argaez_tapia`

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required(Choose One)</td>
<td>Group 1</td>
<td>el_bakry</td>
<td>El-Bakry merit function</td>
</tr>
<tr>
<td>argaez_tapia</td>
<td>The merit function by Tapia and Argaez</td>
<td></td>
<td></td>
</tr>
<tr>
<td>van_shannon</td>
<td>The merit function by Vanderbei and Shanno</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

**el_bakry**

- **Keywords Area**
- **method**
- **optpp_q_newton**
- **merit_function**
- **el_bakry**

El-Bakry merit function
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is $n+1$ function evaluations. For more information, see [20].

argaez_tapia

- Keywords Area
- method
- optpp_q_newton
- merit_function
- argaez_tapia

The merit function by Tapia and Argaez

Specification

Alias: none

Argument(s): none

Description

The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [80].

If the function evaluation is expensive or noisy, set the merit_function to "argaez_tapia" or "van_shanno".

van_shanno

- Keywords Area
- method
- optpp_q_newton
- merit_function
- van_shanno

The merit function by Vanderbei and Shanno
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see[82].

If the function evaluation is expensive or noisy, set the merit function to "argaez_tapia" or "van_shanno".

steplength_to_boundary

- Keywords Area
- method
- optpp_q_newton
- steplength_to_boundary

Controls how close to the boundary of the feasible region the algorithm is allowed to move

Specification

Alias: none
Argument(s): REAL
Default: Merit function dependent: 0.8 (el_bakry), 0.99995 (argaez_tapia), 0.95 (van_shanno)

Description

The steplength_to_boundary specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the el_bakry, argaez_tapia, and van_shanno merit functions, respectively.

centering_parameter

- Keywords Area
- method
- optpp_q_newton
- centering_parameter

Controls how closely the algorithm should follow the "central path"
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.2 (el_bakry), 0.2 (argaez_tapia), 0.1 (van_shanno)

Description

The centering parameter specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the "central path". See[88] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the el_bakry, argaez_tapia, and van_shanno merit functions, respectively.

max_step

- Keywords Area
- method
- optpp_q_newton
- max_step

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The max_step control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The gradient_tolerance control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The gradient_tolerance control is defined for all gradient-based optimizers.

gradient_tolerance

- Keywords Area
- method
- optpp_q_newton
- gradient_tolerance

Stopping criteria based on L2 norm of gradient
6.2. METHOD

Specification

Alias: none
  Argument(s): REAL
  Default: 1.e-4

Description

The gradient_tolerance control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The gradient_tolerance control is defined for all gradient-based optimizers.

max_iterations

- Keywords Area
- method
- optpp_q_newton
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
  Argument(s): INTEGER
  Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global{reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- optpp_q_newton
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics
CHAPTER 6. KEYWORDS AREA

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

speculative

- Keywords Area
- method
- optpp_q_newton
- speculative

Compute speculative gradients
6.2. METHOD

Topics
This keyword is related to the topics:

- method
- independent
- controls

Specification
Alias: none

Argument(s): none

Default: no speculation

Description
When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose.

In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

max_function_evaluations

- Keywords Area
- method
- optpp_q_newton
- max_function_evaluations
Stopping criteria based on number of function evaluations

**Topics**
This keyword is related to the topics:

- `method_independent_controls`

**Specification**

*Alias:* none

*Argument(s):* INTEGER

*Default:* 1000

**Description**
The maximum number of function evaluations (default: 1000). See also `max_iterations`.

**scaling**

- `Keywords Area`
- `method`
- `optpp_q_newton`
- `scaling`

Turn on scaling for variables, responses, and constraints

**Topics**
This keyword is related to the topics:

- `method_independent_controls`

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* no scaling

**Description**
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.
Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a _scales keyword, which gives characteristic values
- a _scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. **none** (default) - no scaling, value of *scales keyword is ignored
2. **value** - multiplicative scaling
3. **auto** - automatic scaling
   
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   
   - two-sided bounds scaled into the interval [0,1];
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component
   
   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. **log** - logarithmic scaling
   
   First, any characteristic values from the optional _scales specification are applied. Then logarithm base 10 scaling is applied.
   
   Logarithmic scaling is not available for linear constraints.
   
   When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The _scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding _scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- **none, auto, log** - optional
- **value** - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
Examples

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

`linear_inequality_constraint_matrix`

- Keywords Area
- method
- optpp_q_newton
- `linear_inequality_constraint_matrix`

Define coefficients of the linear inequality constraints

Topics

This keyword is related to the topics:

- `linear_constraints`

Specification

Alias: none

**Argument(s):** REALLIST

**Default:** no linear inequality constraints

Description

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by `linear_inequality_lower_bounds`, and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$
6.2. METHOD

linear_inequality_lower_bounds

- Keywords Area
- method
- optpp_q_newton
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:
- linear_constraints

Specification

Alias: none

Argument(s): REALIST

Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ (1.e+30, as defined in Minimizer) are treated as $+\text{infinity}$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\text{infinity}$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- optpp_q_newton
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint
Topics
This keyword is related to the topics:

- linear.constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.

Description
In the inequality case, the lower \(a_l\) and upper \(a_u\) bounds provide constraint limits for the two-sided formulation:

\[ a_l \leq Ax \leq a_u \]

Where \(A\) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective.functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[ Ax \leq 0.0 \]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \(+\text{bigRealBoundSize}\) (1.e+30, as defined in Minimizer) are treated as \(+\infty\) and any lower bound values less than \(-\text{bigRealBoundSize}\) are treated as \(-\infty\).

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\text{DBL_MAX}\) \(<\text{-bigRealBoundSize}\)).

linear.inequality.scale_types

- Keywords Area
- method
- optpp_q_newton
- linear.inequality.scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

- linear.constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"
6.2. Method

Description

`linear.inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear.inequality_scales` must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables `x`:

\[
\tilde{x}_j = \frac{x_j^i - x_j^O}{x_j^M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]
\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]
\[
a_L - A_i x_O \leq A_i (\text{diag}(x_M)\tilde{x}) \leq a_U - A_i x_O
\]
\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

`linear.inequality_scales`

- Keywords Area
- method
- optpp_q_newton
- linear.inequality_scales

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- linear.constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1. (no scaling)
CHAPTER 6. KEYWORDS AREA

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
  - 'none' - ignored
  - 'value' - required
  - 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
\begin{align*}
a_L & \leq A_i \tilde{x} \leq a_U \\
a_L & \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O & \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \\
\tilde{a}_L & \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\end{align*}
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

linear_inequality_constraint_matrix

- Keywords Area
  - method
  - optpp_q_newton
  - linear_inequality_constraint_matrix

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none
Argument(s): REALLIST
Default: no linear equality constraints
6.2. METHOD

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

linear_equality_targets
- Keywords Area
- method
- optpp_q_newton
- linear_equality_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
- Argument(s): REALLIST
- Default: vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equality_scale_types
- Keywords Area
- method
- optpp_q_newton
- linear_equality_scale_types

Specify how each linear equality constraint is scaled
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description
linear_equalities_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.
An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_equalities_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}_j = \frac{x_j - x_O}{x_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_equalities_scales

- Keywords Area
- method
- optpp_q_newton
- linear_equalities_scales

Define the characteristic values to scale linear equalities
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constrains

Specification
Alias: none
Argument(s): REALIST
Default: vector values = 1 (no scaling)

Description
Each entry in `linear_equality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.
Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_j = \frac{x_j - x_O}{x_M}
\]

we have the following system for linear inequality constraints

\[
\begin{align*}
\bar{a}_L & \leq A_i \bar{x} \leq \bar{a}_U \\
\bar{a}_L & \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq \bar{a}_U \\
\bar{a}_L - A_i x_O & \leq A_i \text{diag}(x_M)\tilde{x} \leq \bar{a}_U - A_i x_O \\
\bar{a}_L & \leq \tilde{A}_i \bar{x} \leq \bar{a}_U
\end{align*}
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

model_pointer

- Keywords Area
- method
- optpp_q_newton
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```dakota
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
      0.1 0.2 0.6
  sample_type lhs
distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0.0
    upper_bounds = 1.0
  descriptors = 'x1' 'x2'
```
6.2. METHOD

interface
   id_interface = 'I1'
   system async evaluation_concurrency = 5
   analysis_driver = 'text_book'

responses
   response_functions = 3
   no_gradients
   no_hessians

6.2.26 optpp_fd_newton

- Keywords Area
- method
- optpp_fd_newton

Finite Difference Newton optimization method

Topics
This keyword is related to the topics:
- package_optpp
- local_optimization_methods

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>search_method</td>
<td>Select a search method for Newton-based optimizers</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>merit_function</td>
<td>Balance goals of reducing objective function and satisfying constraints</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>steplength_to_boundary</td>
<td>Controls how close to the boundary of the feasible region the algorithm is allowed to move</td>
</tr>
<tr>
<td>Optional</td>
<td>centering_parameter</td>
<td>Controls how closely the algorithm should follow the &quot;central path&quot;</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
<td>------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_step</td>
<td>Max change in design point Stopping criteria based on L2 norm of gradient</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>gradient_tolerance</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Compute speculative gradients Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Turn on scaling for variables, responses, and constraints Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_constraint_matrix</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_lower_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_upper_bounds</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

| Optional | linear_inequality_-scale_types | Specify how each linear inequality constraint is scaled |
| Optional | linear_inequality_-scales | Define the characteristic values to scale linear inequalities |
| Optional | linear_equality_-constraint_matrix | Define coefficients of the linear equalities |
| Optional | linear_equality_-targets | Define target values for the linear equality constraints |
| Optional | linear_equality_-scale_types | Specify how each linear equality constraint is scaled |
| Optional | linear_equality_-scales | Define the characteristic values to scale linear equalities |
| Optional | model_pointer | Identifier for model block to be used by a method |

**Description**

This is a Newton method that expects a gradient and computes a finite-difference approximation to the Hessian. Each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a nonlinear interior-point approach to manage the constraints.

See package_optpp for info related to all optpp methods.

**See Also**

These keywords may also be of interest:

- optpp_cg
- optpp_g_newton
- optpp_pds
- optpp_newton
- optpp_g_newton

**search_method**

- Keywords Area
Select a search method for Newton-based optimizers

**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>value_based_line_search</td>
<td>Use only function values for line search</td>
</tr>
<tr>
<td>gradient_based_line_search</td>
<td></td>
<td></td>
<td>Set the search method to use the gradient</td>
</tr>
<tr>
<td>trust_region</td>
<td></td>
<td></td>
<td>Use trust region as the globalization strategy</td>
</tr>
<tr>
<td>tr_pds</td>
<td></td>
<td></td>
<td>Use direct search as the local search in a trust region method</td>
</tr>
</tbody>
</table>

**Description**

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [61]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_based_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

**value_based_line_search**

- Keywords Area
- method
- optpp_fd_newton
- search_method
6.2. METHOD

- value_based_line_search

Use only function values for line search

Specification

Alias: none
Argument(s): none
Default: trust_region (unconstrained), value_based_line_search (bound/general constraints)

Description

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_base_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

gradient_based_line_search

- Keywords Area
- method
- optpp_fd_newton
- search_method
- gradient_based_line_search

Set the search method to use the gradient

Specification

Alias: none
Argument(s): none

Description

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_base_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods
additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

**trust_region**

- Keywords Area
- method
- optpp_fd_newton
- search_method
- trust_region

Use trust region as the globalization strategy.

**Specification**

*Alias*: none  
*Argument(s)*: none

**Description**

The trust_region optional group specification can be used to specify the initial size of the trust region (using initial_size) relative to the total variable bounds, the minimum size of the trust region (using minimum_size), the contraction factor for the trust region size (using contraction_factor) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using expansion_factor) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using contract_threshold) and the trust region size expansion threshold (using expand_threshold). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command contract_threshold sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command expand_threshold determines the trust region value above which the trust region will expand for the next SBL iteration.

**tr_pds**

- Keywords Area
- method
- optpp_fd_newton
- search_method
- tr_pds

Use direct search as the local search in a trust region method.
6.2. METHOD

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [61]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_based_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

**merit_function**

- **Keywords Area**
  - `method`
  - `optpp_fd_newton`
  - `merit_function`

  Balance goals of reducing objective function and satisfying constraints

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** 

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td><code>argaez_tapia</code></td>
<td>El-Bakry merit function</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>van_shanno</code></td>
<td>The merit function by Vanderbei and Shanno</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>argaez_tapia</code></td>
<td>The merit function by Tapia and Argaez</td>
</tr>
</tbody>
</table>

**Description**

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.
el_bakry

- Keywords Area
- method
- optpp_fd_newton
- merit_function
- el_bakry

El-Bakry merit function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is n+1 function evaluations. For more information, see [20].

argaez_tapia

- Keywords Area
- method
- optpp_fd_newton
- merit_function
- argaez_tapia

The merit function by Tapia and Argaez

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [80].

If the function evaluation is expensive or noisy, set the **merit_function** to "argaez_tapia" or "van_shanno".
6.2. **METHOD**

**van_shanno**

- Keywords Area
- method
- optpp_fd_newton
- merit_function
- van_shanno

The merit function by Vanderbei and Shanno

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see[82].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

**steplength_to_boundary**

- Keywords Area
- method
- optpp_fd_newton
- steplength_to_boundary

Controls how close to the boundary of the feasible region the algorithm is allowed to move

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** Merit function dependent: 0.8 (el_bakry), 0.99995 (argaez_tapia), 0.95 (van_shanno)

**Description**

The `steplength_to_boundary` specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the el_bakry, argaez_tapia, and van_shanno merit functions, respectively.
centering parameter

- Keywords Area
- method
- optpp_fd_newton
- centering parameter

Controls how closely the algorithm should follow the "central path"

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.2 (el_bakry), 0.2 (argaez_tapia), 0.1 (van_shanno)

Description

The centering parameter specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the "central path". See[88] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the el_bakry, argaez_tapia, and van_shanno merit functions, respectively.

max_step

- Keywords Area
- method
- optpp_fd_newton
- max_step

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The max_step control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The gradient_tolerance control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The gradient_tolerance control is defined for all gradient-based optimizers.
6.2. METHOD

gradient_tolerance

- Keywords Area
- method
- optpp_fd_newton
- gradient_tolerance

Stopping criteria based on L2 norm of gradient

Specification

Alias: none

Argument(s): REAL
Default: 1.e-4

Description

The gradient_tolerance control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The gradient_tolerance control is defined for all gradient-based optimizers.

max_iterations

- Keywords Area
- method
- optpp_fd_newton
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.
CHAPTER 6. KEYWORDS AREA

convergence_tolerance
  • Keywords Area
  • method
  • optpp_fd_newton
  • convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:
  • method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: 1.e-4

Description
The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration. Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library
This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:
  • DOT: must be satisfied for two consecutive iterations
  • NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
  • NL2SOL: See nl2sol
speculative

- Keywords Area
- method
- optpp_fd_newton
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none
Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable
only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

max_function_evaluations

- Keywords Area
- method
- optpp_fd_newton
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling

- Keywords Area
- method
- optpp_fd_newton
- scaling

Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling
Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling
   - First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
     - two-sided bounds scaled into the interval [0,1];
     - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
     - no bounds or targets: no automatic scaling possible, therefore no scaling for this component
   - Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. log - logarithmic scaling
   - First, any characteristic values from the optional _scales specification are applied. Then logarithm base 10 scaling is applied.
   - Logarithmic scaling is not available for linear constraints.
   - When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above. Depending on the scale type, the characteristic values may be required or optional.
• none, auto, log - optional
• value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0e10 \times \text{DBL_MIN}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples
The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

linear_inequality_constraint_matrix

• Keywords Area
• method
• optpp_fd_newton
• linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics

This keyword is related to the topics:

• linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear inequality constraints

Description

In the inequality case, the constraint matrix \(A\) provides coefficients for the variables in the two-sided formulation:

\[ a_l \leq Ax \leq a_u \]

Where the bounds are optionally specified by `linear_inequality_lower_bounds`, and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

\[ Ax \leq 0.0 \]
linear_inequality_lower_bounds

- Keywords Area
- method
- optpp_fd_newton
- linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = -infinity

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_upper_bounds

- Keywords Area
- method
- optpp_fd_newton
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint
Topics
This keyword is related to the topics:
  • linear_constraints

Specification
Alias: none
  Argument(s): REALLIST
  Default: vector values = 0.

Description
In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
al \leq Ax \leq au
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).

linear_inequality_scale_types
  • Keywords Area
  • method
  • optpp_fd_newton
  • linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:
  • linear_constraints

Specification
Alias: none
  Argument(s): STRINGLIST
  Default: vector values = "none"
Description

`linear.inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear.inequality_scales` must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_j^i = \frac{x_j^i - x_{jO}}{x_{jM}}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

`linear.inequality_scales`

- **Keywords Area**
- **method**
- **optpp_fd_newton**
- **linear.inequality_scales**

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- **linear.constraints**

Specification

Alias: none

**Argument(s):** REALLIST

**Default:** vector values = 1. (no scaling)
CHAPTER 6. KEYWORDS AREA

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
  - 'none' - ignored
  - 'value' - required
  - 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables x:

\[
\tilde{x}_j = \frac{x_j^O - x_j^O}{x_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_equality_constraint_matrix

- Keywords Area
- method
- optpp_fd_newton
- linear_equality_constraint_matrix

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints
6.2. METHOD

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

**linear Equality Targets**

- Keywords Area
- method
- optpp_fd_newton
- linear equality_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:

- linear constraints

Specification

Alias: none

**Argument(s):** REALIST

**Default:** vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0.0$$

**linear equality_scale_types**

- Keywords Area
- method
- optpp_fd_newton
- linear equality_scale_types

Specify how each linear equality constraint is scaled
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:
- linear constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description
linear.equality_scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.
An entry may be selected for each constraint. The options are:
- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear.equality.scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}_j = \frac{x_j - x_{O}}{x_{M}}$$
we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear.equality.scales
- Keywords Area
- method
- optpp_fd_newton
- linear.equality.scales

Define the characteristic values to scale linear equalities
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 1. (no scaling)

Description
Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword. Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}
\]

we have the following system for linear inequality constraints

\[
\begin{align*}
a_L & \leq A_i x \leq a_U \\
a_L & \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O & \leq A_i (\text{diag}(x_M) \tilde{x} - A_i x_O) \leq a_U - A_i x_O \\
\tilde{a}_L & \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\end{align*}
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

model_pointer

- Keywords Area
- method
- optpp_fd_newton
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = ‘UQ’
  model_pointer = ‘SURR’
  sampling,
    samples = 10
    seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
  0.1 0.2 0.6
  0.1 0.2 0.6
  sample_type lhs
distribution cumulative

model
  id_model = ‘SURR’
  surrogate global,
  dace_method_pointer = ‘DACE’
  polynomial quadratic

method
  id_method = ‘DACE’
  model_pointer = ‘DACE_M’
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = ‘DACE_M’
  single
  interface_pointer = ’I’

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = ‘x1’ ‘x2’
```
6.2. METHOD

interface
id_interface = 'I1'

system async evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.27 optpp_g_newton

- Keywords Area
  - method
  - optpp_g_newton

Newton method based least-squares calibration

Topics
This keyword is related to the topics:
  - package_optpp
  - local_optimization_methods

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>search_method</td>
<td></td>
<td>Select a search method for Newton-based optimizers</td>
</tr>
<tr>
<td>Optional</td>
<td>merit_function</td>
<td></td>
<td>Balance goals of reducing objective function and satisfying constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>steplength_to_boundary</td>
<td></td>
<td>Controls how close to the boundary of the feasible region the algorithm is allowed to move</td>
</tr>
<tr>
<td>Optional</td>
<td>centering parameter</td>
<td>Controls how closely the algorithm should follow the &quot;central path&quot;</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------</td>
<td>---------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_step</td>
<td>Max change in design point</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>gradient tolerance</td>
<td>Stopping criteria based on L2 norm of gradient</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>convergence tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>speculative</td>
<td>Compute speculative gradients</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

| Optional | linear_inequality_.scale_types | Specify how each linear inequality constraint is scaled |
| Optional | linear_inequality_.scales | Define the characteristic values to scale linear inequalities |
| Optional | linear_inequality_.constraint_matrix | Define coefficients of the linear equalities |
| Optional | linear_equality_.targets | Define target values for the linear equality constraints |
| Optional | linear_equality_.scale_types | Specify how each linear equality constraint is scaled |
| Optional | linear_equality_.scales | Define the characteristic values to scale linear equalities |
| Optional | model_pointer | Identifier for model block to be used by a method |

Description

The Gauss-Newton algorithm is available as `optpp_g.newton` and supports unconstrained, bound-constrained, and generally-constrained problems. When interfaced with the unconstrained, bound-constrained, and nonlinear interior point full-Newton optimizers from the OPT++ library, it provides a Gauss-Newton least squares capability which – on zero-residual test problems – can exhibit quadratic convergence rates near the solution. (Real problems almost never have zero residuals, i.e., perfect fits.)

See package `optpp` for info related to all `optpp` methods.

See Also

These keywords may also be of interest:

- `optpp_cg`
- `optpp_pds`
- `optpp_fd_newton`
- `optpp_newton`
- `optpp_g_newton`
- `field_calibration_terms`
search_method

- Keywords Area
- method
- optpp_g_newton
- search_method

Select a search method for Newton-based optimizers

**Specification**

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>value_based_line_search</td>
<td>Use only function values for line search</td>
</tr>
<tr>
<td>gradient_based_line_search</td>
<td></td>
<td>Set the search method to use the gradient</td>
<td></td>
</tr>
<tr>
<td>trust_region</td>
<td></td>
<td>Use trust region as the globalization strategy</td>
<td></td>
</tr>
<tr>
<td>tr_pds</td>
<td></td>
<td>Use direct search as the local search in a trust region method</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by [61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_based_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

value_based_line_search

- Keywords Area
- method
6.2. METHOD

- `optpp_g_newton`
- `search_method`
- `value_based_line_search`

Use only function values for line search

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** trust_region (unconstrained), value_based_line_search (bound/general constraints)

**Description**

The `search_method` control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_based_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

**gradient_based_line_search**

- **Keywords Area**

- **method**

- `optpp_g_newton`

- `search_method`

- `gradient_based_line_search`

Set the search method to use the gradient

**Specification**

**Alias:** none

**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_base_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

trust_region

- Keywords Area
- method
- optpp.g_newton
- search_method
- trust_region

Use trust region as the globalization strategy.

Specification

Alias: none
Argument(s): none

Description

The trust_region optional group specification can be used to specify the initial size of the trust region (using initial_size) relative to the total variable bounds, the minimum size of the trust region (using minimum_size), the contraction factor for the trust region size (using contraction_factor) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using expansion_factor) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using contract_threshold) and the trust region size expansion threshold (using expand_threshold). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command contract_threshold sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command expand_threshold determines the trust region value above which the trust region will expand for the next SBL iteration.
6.2. METHOD

- optpp.g_newton
- search_method
- tr.pds

Use direct search as the local search in a trust region method

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `search_method` control is defined for all Newton-based optimizers and is used to select between trust-region, gradient-based line search, and value-based line search methods. The `gradient_based_line_search` option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_based_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr.pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

**merit_function**

- **Keywords Area**
- method
- `optpp.g_newton`
- `merit_function`

Balance goals of reducing objective function and satisfying constraints

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** argaez_tapia

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required (Choose One)</strong></td>
<td><strong>Group 1</strong></td>
<td>el_bakry</td>
<td>El-Bakry merit function</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>argaez_tapia</td>
<td>The merit function by Tapia and Argaez</td>
</tr>
<tr>
<td>van_shanno</td>
<td>The merit function by Vanderbei and Shanno</td>
</tr>
</tbody>
</table>

Description

A merit function is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

el_bakry

- Keywords Area
- method
- optpp_g_newton
- merit_function
- el_bakry

El-Bakry merit function

Specification

Alias: none

Argument(s): none

Description

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is n+1 function evaluations. For more information, see [20].

argaez_tapia

- Keywords Area
- method
- optpp_g_newton
- merit_function
- argaez_tapia

The merit function by Tapia and Argaez

Specification

Alias: none

Argument(s): none
6.2. METHOD

Description
The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [80].

If the function evaluation is expensive or noisy, set the merit_function to "argaez_tapia" or "van_shanno".

van_shanno

- Keywords Area
  - method
  - optpp_g_newton
  - merit_function
  - van_shanno

The merit function by Vanderbei and Shanno

Specification
Alias: none
Argument(s): none

Description
The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see [82].

If the function evaluation is expensive or noisy, set the merit_function to "argaez_tapia" or "van_shanno".

steplength_to_boundary

- Keywords Area
  - method
  - optpp_g_newton
  - steplength_to_boundary

Controls how close to the boundary of the feasible region the algorithm is allowed to move

Specification
Alias: none
Argument(s): REAL
Default: Merit function dependent: 0.8 (el_bakry), 0.99995 (argaez_tapia), 0.95 (van_shanno)
CHAPTER 6. KEYWORDS AREA

Description

The `steplength_to_boundary` specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

centering parameter

- Keywords Area
- method
- `optpp.g_newton`
- `centering_parameter`

Controls how closely the algorithm should follow the “central path”

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.2 (`el_bakry`), 0.2 (`argaez_tapia`), 0.1 (`van_shanno`)

Description

The `centering_parameters` specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the “central path”. See[88] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

max_step

- Keywords Area
- method
- `optpp.g_newton`
- `max_step`

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.
6.2. **METHOD**

**Description**

The *max_step* control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The *gradient_tolerance* control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The *gradient_tolerance* control is defined for all gradient-based optimizers.

**gradient_tolerance**

- **Keywords Area**
  - method
  - optpp_g_newton
  - gradient_tolerance

  Stopping criteria based on L2 norm of gradient

**Specification**

Alias: none

Argument(s): REAL

Default: 1.e-4

**Description**

The *gradient_tolerance* control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The *gradient_tolerance* control is defined for all gradient-based optimizers.

**max_iterations**

- **Keywords Area**
  - method
  - optpp_g_newton
  - max_iterations

  Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_[reliability, interval_est, evidence] / efficient_global: 25+n)

Description

The maximum number of iterations (default: 100). See also max.function_evaluations.

convergence_tolerance

• Keywords Area
• method
• optpp_g_newton
• convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

• method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

• DOT: must be satisfied for two consecutive iterations
6.2. METHOD

- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

- NL2SOL: See nl2sol

speculative

- Keywords Area
- method
- optpp_g_newton
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose.
In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- optpp_g_newton
- max_function_evaluations

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

- Argument(s): INTEGER
- Default: 1000

**Description**

The maximum number of function evaluations (default: 1000). See also `max_iterations`.

**scaling**

- Keywords Area
- method
- optpp_g_newton
- scaling

Turn on scaling for variables, responses, and constraints
6.2. METHOD

Topics
This keyword is related to the topics:

- method-independent_controls

Specification

Alias: none
- Argument(s): none
- Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. **log** - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The `*_scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*_scale_type`, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- **none**, **auto**, **log** - optional
- **value** - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10 DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

**linear_inequality_constraint_matrix**

- **Keywords Area**
- **method**
- **optpp.g.newton**
- **linear_inequality_constraint_matrix**

Define coefficients of the linear inequality constraints

**Topics**

This keyword is related to the topics:

- **linear_constraints**
6.2. METHOD

Specification

Alias: none
   Argument(s): REALLIST
   Default: no linear inequality constraints

Description

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by `linear_inequality_lower_bounds`, and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

linear_inequality_lower_bounds

- Keywords Area
- method
- `optpp_g_newton`
- `linear_inequality_lower_bounds`

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- `linear_constraints`

Specification

Alias: none
   Argument(s): REALLIST
   Default: vector values = -infinity

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see `objective_functions`), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$
result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).

**linear_inequality_upper_bounds**

- Keywords Area
- method
- optpp_g_newton
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

*Alias*: none

*Argument(s)*: REALLIST

*Default*: vector values = 0.

**Description**

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).
6.2. METHOD

linear_inequality_scale_types

- Keywords Area
- method
- optpp_g_newton
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

linear_inequality_scale_types provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear_inequality_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$
\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}
$$

we have the following system for linear inequality constraints

$$
a_L \leq A_i x \leq a_U$$
$$
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$
$$
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$
a_L \leq \tilde{A}_i \tilde{x} \leq a_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].
linear_inequality_scales

- Keywords Area
- method
- optpp_g_newton
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

Description

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`
- 'none' - ignored
- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].
6.2. METHOD

linear.equality.constraint_matrix
- Keywords Area
- method
- optpp.g.newton
- linear.equality.constraint_matrix

Define coefficients of the linear equalities

Topics
This keyword is related to the topics:
- linear.constraints

Specification
Alias: none
Argument(s): REALLIST
Default: no linear equality constraints

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = at$$

linear.equality.targets
- Keywords Area
- method
- optpp.g.newton
- linear.equality.targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:
- linear.constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.
Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0$$

linear.equality.scale_types

- Keywords Area
- method
- optpp.g.newton
- linear.equality.scale_types

Specify how each linear equality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = “none”

Description

linear.equality.scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ‘none’ - no scaling
- ‘value’ - characteristic value if this is chosen, then linear.equality.scales must be specified
- ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_t x \leq a_U$$
6.2. METHOD

\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i (\text{diag}(x_M) \tilde{x}) \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_equality_scales**

- Keywords Area
- method
- optpp_g_newton
- linear_equality_scales

Define the characteristic values to scale linear equalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = 1. (no scaling)

**Description**

Each entry in `linear_equality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the `method` section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[ \tilde{x}^j = \frac{x^j - x_O^j}{x_M^j} \]

we have the following system for linear inequality constraints

\[ a_L \leq A_i x \leq a_U \]
\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i (\text{diag}(x_M) \tilde{x}) \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].
model_pointer

- Keywords Area
- method
- optpp.g_newton
- model_pointer

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- block_pointer

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

**Examples**

```
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
```


6.2. METHOD

```
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians
```

6.2.28 optpp.newton

- Keywords Area
- method
  - optpp.newton

Newton method based optimization

Topics

This keyword is related to the topics:

- package.optpp
- local.optimization.methods

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{search_method}</td>
<td>Select a search method for Newton-based optimizers</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{merit_function}</td>
<td>Balance goals of reducing objective function and satisfying constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{steplength_to_boundary}</td>
<td>Controls how close to the boundary of the feasible region the algorithm is allowed to move</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{centering_parameter}</td>
<td>Controls how closely the algorithm should follow the &quot;central path&quot;</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{max_step}</td>
<td>Max change in design point</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{gradient_tolerance}</td>
<td>Stopping criteria based on L2 norm of gradient</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{max_iterations}</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{convergence_tolerance}</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{speculative}</td>
<td>Compute speculative gradients</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>\textit{max_function_evaluations}</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
</tbody>
</table>
6.2. **METHOD**

<table>
<thead>
<tr>
<th>Optional</th>
<th>scaling</th>
<th>Turn on scaling for variables, responses, and constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>linear_inequality_-</td>
<td>Define coefficients of the linear inequality constraints</td>
</tr>
<tr>
<td></td>
<td>constraint_matrix</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-</td>
<td>Define lower bounds for the linear inequality constraint</td>
</tr>
<tr>
<td></td>
<td>lower_bounds</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-</td>
<td>Define upper bounds for the linear inequality constraint</td>
</tr>
<tr>
<td></td>
<td>upper_bounds</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-</td>
<td>Specify how each linear inequality constraint is scaled</td>
</tr>
<tr>
<td></td>
<td>scale_types</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-</td>
<td>Define the characteristic values to scale linear inequalities</td>
</tr>
<tr>
<td></td>
<td>scales</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-</td>
<td>Define coefficients of the linear equalities</td>
</tr>
<tr>
<td></td>
<td>constraint_matrix</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-</td>
<td>Define target values for the linear equality constraints</td>
</tr>
<tr>
<td></td>
<td>targets</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-</td>
<td>Specify how each linear equality constraint is scaled</td>
</tr>
<tr>
<td></td>
<td>scale_types</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-</td>
<td>Define the characteristic values to scale linear equalities</td>
</tr>
<tr>
<td></td>
<td>scales</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

This is a full Newton method that expects a gradient and a Hessian. Each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a non-linear interior-point approach to manage the constraints.

See package_optpp for info related to all optpp methods.
CHAPTER 6. KEYWORDS AREA

See Also

These keywords may also be of interest:

- optpp.cg
- optpp.g_newton
- optpp.pds
- optpp.fd_newton
- optpp.g_newton

search_method

- Keywords Area
- method
- optpp.newton
- search_method

Select a search method for Newton-based optimizers

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>[Group 1]</td>
<td>value_based_line_search</td>
<td>Use only function values for line search</td>
</tr>
<tr>
<td>gradient_based_line_search</td>
<td>Set the search method to use the gradient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trust_region</td>
<td>Use trust region as the globalization strategy.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tr_pds</td>
<td>Use direct search as the local search in a trust region method</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_base_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function
and gradient at the trial point. Consequently, given expensive function evaluations, the \texttt{value\_based\_line\_search} method is preferred to the \texttt{gradient\_based\_line\_search} method. Each of these Newton methods additionally supports the \texttt{tr\_pds} selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a \texttt{trust\_region} search method is the default for unconstrained problems.

\texttt{value\_based\_line\_search}

- Keywords Area
- method
- optpp\_newton
- search\_method
- value\_based\_line\_search

Use only function values for line search

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** trust\_region (unconstrained), value\_based\_line\_search (bound/general constraints)

**Description**

The search\_method control is defined for all Newton-based optimizers and is used to select between trust\_region, gradient\_based\_line\_search, and value\_based\_line\_search methods. The gradient\_based\_line\_search option uses the line search method proposed by\cite{61}. This option satisfies sufficient decrease and curvature conditions; whereas, value\_based\_line\_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient\_based\_line\_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value\_based\_line\_search method is preferred to the gradient\_based\_line\_search method. Each of these Newton methods additionally supports the tr\_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust\_region search method is the default for unconstrained problems.

\texttt{gradient\_based\_line\_search}

- Keywords Area
- method
- optpp\_newton
- search\_method
- gradient\_based\_line\_search

Set the search method to use the gradient
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none

Description

The search method control is defined for all Newton-based optimizers and is used to select between trust-region, gradient-based line search, and value-based line search methods. The gradient-based line search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_based_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

trust_region

- Keywords Area
- method
- optpp_newton
- search_method
- trust_region

Use trust region as the globalization strategy.

Specification

Alias: none
Argument(s): none

Description

The trust_region optional group specification can be used to specify the initial size of the trust region (using initial_size) relative to the total variable bounds, the minimum size of the trust region (using minimum_size), the contraction factor for the trust region size (using contraction_factor) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using expansion_factor) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using contract_threshold) and the trust region size expansion threshold (using expand_threshold). These two commands are related to what is called the trust region ratio, which is the actual decrease in the trust model divided by the predicted decrease in the trust model in the current trust region. The command contract_threshold sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command expand_threshold determines the trust region value above which the trust region will expand for the next SBL iteration.
6.2. METHOD

tr_pds

- Keywords Area
- method
- optpp_newton
- search_method
- tr_pds

Use direct search as the local search in a trust region method

Specification

Alias: none
Argument(s): none

Description

The search_method control is defined for all Newton-based optimizers and is used to select between trust_region, gradient_based_line_search, and value_based_line_search methods. The gradient_based_line_search option uses the line search method proposed by[61]. This option satisfies sufficient decrease and curvature conditions; whereas, value_base_line_search only satisfies the sufficient decrease condition. At each line search iteration, the gradient_based_line_search method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the value_based_line_search method is preferred to the gradient_based_line_search method. Each of these Newton methods additionally supports the tr_pds selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a trust_region search method is the default for unconstrained problems.

merit_function

- Keywords Area
- method
- optpp_newton
- merit_function

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none
Argument(s): none
Default: argaez_tapia
### Description

A **merit function** is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

**el_bakry**

- **Keywords Area**
- **method**
- **optpp.newton**
- **merit.function**
- **el_bakry**

El-Bakry merit function

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is $n+1$ function evaluations. For more information, see[20].

**argaez_tapia**

- **Keywords Area**
- **method**
- **optpp.newton**
- **merit.function**
- **argaez_tapia**

The merit function by Tapia and Argaez
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [80].

If the function evaluation is expensive or noisy, set the merit function to "argaez_tapia" or "van_shanno".

van_shanno

- Keywords Area
- method
- optpp_newton
- merit_function
- van_shanno

The merit function by Vanderbei and Shanno

Specification

Alias: none
Argument(s): none

Description

The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see [82].

If the function evaluation is expensive or noisy, set the merit function to "argaez_tapia" or "van_shanno".

steplength_to_boundary

- Keywords Area
- method
- optpp_newton
- steplength_to_boundary

Controls how close to the boundary of the feasible region the algorithm is allowed to move
CHAPTER 6. KEYWORDS AREA

Specification

**Alias:** none

**Argument(s):** REAL

**Default:** Merit function dependent: 0.8 (el_bakry), 0.99995 (argaez_tapia), 0.95 (van_shanno)

**Description**

The `steplength_to_boundary` specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the el_bakry, argaez_tapia, and van_shanno merit functions, respectively.

centering_parameter

- **Keywords Area**
- **method**
- `optpp_newton`
- **centering_parameter**

Controls how closely the algorithm should follow the "central path"

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** Merit function dependent: 0.2 (el_bakry), 0.2 (argaez_tapia), 0.1 (van_shanno)

**Description**

The `centering_parameter` specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the "central path". See[88] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the el_bakry, argaez_tapia, and van_shanno merit functions, respectively.

max_step

- **Keywords Area**
- **method**
- `optpp_newton`
- **max_step**

Max change in design point
6.2. **METHOD**

**Specification**

Alias: none

Argument(s): REAL

Default: 1000.

**Description**

The max_step control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The gradient_tolerance control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The gradient_tolerance control is defined for all gradient-based optimizers.

**gradient_tolerance**

- Keywords Area
- method
- optpp_newton
- gradient_tolerance

Stopping critera based on L2 norm of gradient

**Specification**

Alias: none

Argument(s): REAL

Default: 1.e-4

**Description**

The gradient_tolerance control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The gradient_tolerance control is defined for all gradient-based optimizers.

**max_iterations**

- Keywords Area
- method
- optpp_newton
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_{global: 25+n})

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- optpp_newton
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
6.2. METHOD

- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

- NL2SOL: See nl2sol

speculative

- Keywords Area
- method
- optpp_newton
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none
Argument(s): none
Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose.
In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

**max_function_evaluations**

- Keywords Area
- method
- optpp_newton
- max_function_evaluations

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

Argument(s): INTEGER

Default: 1000

**Description**

The maximum number of function evaluations (default: 1000). See also max_iterations.

**scaling**

- Keywords Area
- method
- optpp_newton
- scaling

Turn on scaling for variables, responses, and constraints
6.2. METHOD

Topics
This keyword is related to the topics:

• method-independent-controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all .scale_types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

• a *scales keyword, which gives characteristic values
• a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

• two-sided bounds scaled into the interval [0,1];
• one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
• no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Futher, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

• none, auto, log - optional

• value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

linear_inequality_constraint_matrix

• Keywords Area

• method

• optpp_newton

• linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics

This keyword is related to the topics:

• linear_constraints
6.2. METHOD

Specification

Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

Description

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by `linear_inequality_lower_bounds`, and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to $-\infty$, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

`linear_inequality_lower_bounds`

- Keywords Area
- method
- optpp_newton
- `linear_inequality_lower_bounds`

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- `linear_constraints`

Specification

Alias: none
Argument(s): REALLIST
Default: vector values = $-\infty$

Description

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see `objective_functions`), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$
result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \( +\text{bigRealBoundSize} \) (1.e+30, as defined in Minimizer) are treated as \( +\infty \) and any lower bound values less than \( -\text{bigRealBoundSize} \) are treated as \( -\infty \).

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\text{DBL_MAX} < -\text{bigRealBoundSize}\)).

\textbf{linear\_inequality\_upper\_bounds}

- Keywords Area
- method
- optpp\_newton
- linear\_inequality\_upper\_bounds

Define upper bounds for the linear inequality constraint

\textbf{Topics}

This keyword is related to the topics:

- linear\_constraints

\textbf{Specification}

Alias: none

Argument(s): REALLIST

Default: vector values = 0.

\textbf{Description}

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective\_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than \( +\text{bigRealBoundSize} \) (1.e+30, as defined in Minimizer) are treated as \( +\infty \) and any lower bound values less than \( -\text{bigRealBoundSize} \) are treated as \( -\infty \).

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\text{DBL_MAX} < -\text{bigRealBoundSize}\)).
6.2. METHOD

linear_inequality_scale_types

- Keywords Area
- method
- optpp_newton
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

`linear_inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- ‘none’ - no scaling
- ‘value’ - characteristic value if this is chosen, then `linear_inequality_scales` must be specified
- ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

\[ \tilde{x}^j = \frac{x^j - x_O^j}{x_M^j} \]

we have the following system for linear inequality constraints

\[ a_L \leq A_i x \leq a_U \]
\[ a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$. 
linear_inequality_scales

- Keywords Area
- method
- optpp.newton
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

Topics
This keyword is related to the topics:
- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
  - 'none' - ignored
  - 'value' - required
  - 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}_j = \frac{x_j^O - x_j}{x_M}
\]

we have the following system for linear inequality constraints

\[
\begin{align*}
a_L & \leq A_i \tilde{x} \leq a_U \\
A_i & \leq A_i \left( \text{diag}(x_M) \tilde{x} + x_O \right) \leq a_U \\
a_L - A_i x_O & \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O \\
\tilde{a}_L & \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\end{align*}
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).
6.2. METHOD

linear_equation_constraint_matrix

- Keywords Area
- method
- optpp_newton
- linear_equation_constraint_matrix

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST
Default: no linear equality constraints

Description

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

linear_equation_targets

- Keywords Area
- method
- optpp_newton
- linear_equation_targets

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST
Default: vector values = 0.
Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear.equality.scale_types
- Keywords Area
- method
- optpp_newton
- linear.equality.scale_types

Specify how each linear equality constraint is scaled

Topics
This keyword is related to the topics:
- linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description
linear.equality.scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:
- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear.equality.scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_t x \leq a_U$$
6.2. METHOD

\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i (\text{diag}(x_M) \tilde{x}) \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_inequality_scales**

- **Keywords Area**
- **method**
- **optpp_newton**
- **linear_inequality_scales**

Define the characteristic values to scale linear equalities

**Topics**

This keyword is related to the topics:

- **linear_constraints**

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = 1. (no scaling)

**Description**

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the `method` section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables \( x \):

\[ \tilde{x}^j = \frac{x^j - x_O^j}{x_M^j} \]

we have the following system for linear inequality constraints

\[ a_L \leq A_i x \leq a_U \]
\[ a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U \]
\[ a_L - A_i x_O \leq A_i (\text{diag}(x_M) \tilde{x}) \leq a_U - A_i x_O \]
\[ \tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].
CHAPTER 6. KEYWORDS AREA

model_pointer

- Keywords Area
- method
- optpp_newton
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:
- block_pointer

Specification

Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.
See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.29 optpp_pds

- Keywords Area
- method
- optpp_pds

Simplex-based derivative free optimization method

Topics

This keyword is related to the topics:
- package_optpp

Specification

Alias: none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


### Optional Keywords Area

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>search_scheme_size</code></td>
<td>Number of points to be used in the direct search template</td>
</tr>
<tr>
<td><code>max_iterations</code></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td><code>convergence_tolerance</code></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td><code>max_function_evaluations</code></td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
<tr>
<td><code>scaling</code></td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td><code>model_pointer</code></td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

### Description

The direct search algorithm, PDS (parallel direct search method), supports bound constraints. The PDS method can directly exploit asynchronous evaluations; however, this capability has not yet been implemented in Dakota.

See package_optpp for info related to all optpp methods.

### See Also

These keywords may also be of interest:

- `optpp_cg`
- `optpp_g_newton`
- `optpp_fd_newton`
- `optpp_newton`
- `optpp_g_newton`

**search_scheme_size**

- Keywords Area
6.2. METHOD

- method
- optpp_pds
- search_scheme_size

Number of points to be used in the direct search template

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** 32

**Description**

The search_scheme_size is defined for the PDS method to specify the number of points to be used in the direct search template.

**max_iterations**

- Keywords Area
- method
- optpp_pds
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also max_function_evaluations.
convergence_tolerance

- Keywords Area
- method
- optpp.pds
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** 1.e-4

**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol
max_function_evaluations

- **Keywords Area**
- **method**
- **optpp_pds**
- **max_function_evaluations**

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 1000

**Description**

The maximum number of function evaluations (default: 1000). See also `max_iterations`.

**scaling**

- **Keywords Area**
- **method**
- **optpp_pds**
- **scaling**

Turn on scaling for variables, responses, and constraints

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no scaling
Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the `scaling` keyword is omitted, all `_scale_type` and `*_scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `*_scales` keyword, which gives characteristic values
- a `*_scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. `none` (default) - no scaling, value of `*_scales` keyword is ignored
2. `value` - multiplicative scaling
3. `auto` - automatic scaling
4. `log` - logarithmic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*_scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*_scale_type`, as described above.

Depending on the scale type, the characteristic values may be required or optional.
6.2. METHOD

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0 \times 10^{10}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples
The two examples below are equivalent:

```
responses
objective_functions 3
  sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
  sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

model_pointer

- Keywords Area
- method
- optpp.pds
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description
The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.
**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians
```

### 6.2.30 asynch_pattern_search

- Keywords Area
6.2. METHOD

- method
- asynch_pattern_search

Pattern search, derivative free optimization method

Topics
This keyword is related to the topics:

- package_hopspack
- global_optimization_methods

Specification
Alias: coliny_apps
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>initial_delta</td>
<td>Initial step size for non-gradient based optimizers</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>contraction_factor</td>
<td>Amount by which step length is rescaled</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>threshold_delta</td>
<td>Stopping criteria based on step length or pattern size</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>solution_target</td>
<td>Stopping criteria based on objective function value</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>synchronization</td>
<td>Select how Dakota schedules function evaluations in a pattern search</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>merit_function</td>
<td>Balance goals of reducing objective function and satisfying constraints</td>
</tr>
<tr>
<td>Keyphrase</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>constraint_penalty</code></td>
<td>Multiplier for the penalty function Smoothing value for smoothed penalty functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>smoothing_factor</code></td>
<td>Define coefficients of the linear inequality constraints Define lower bounds for the linear inequality constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_inequality_constraint_matrix</code></td>
<td>Define coefficients of the linear inequality constraints Specify how each linear inequality constraint is scaled</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_inequality_lower_bounds</code></td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_inequality_upper_bounds</code></td>
<td>Specify how each linear inequality constraint is scaled</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_inequality_scale_types</code></td>
<td>Define the characteristic values to scale linear inequalities</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_inequality_scales</code></td>
<td>Define coefficients of the linear equalities</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_equation_constraint_matrix</code></td>
<td>Define target values for the linear equality constraints Specify how each linear equality constraint is scaled</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_equation_targets</code></td>
<td>Define the characteristic values to scale linear equalities</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_equation_scale_types</code></td>
<td>Define the characteristic values to scale linear equalities</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>linear_equation_scales</code></td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>constraint_tolerance</code></td>
<td>The maximum allowable value of constraint violation still considered to be feasible</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Option</th>
<th>max_function_evaluations</th>
<th>Stopping criteria based on number of function evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

Description

The asynchronous parallel pattern search (APPS) algorithm [37] is a fully asynchronous pattern search technique in that the search along each offset direction continues without waiting for searches along other directions to finish.

Currently, APPS only supports coordinate bases with a total of $2n$ function evaluations in the pattern, and these patterns may only contract.

Concurrency

APPS exploits parallelism through the use of Dakota’s concurrent function evaluations. The variant of the algorithm that is currently exposed, however, limits the amount of concurrency that can be exploited. In particular, APPS can leverage an evaluation concurrency level of at most twice the number of variables. More options that allow for greater evaluation concurrency may be exposed in future releases.

Algorithm Behavior

- **initial_delta**: the initial step length, must be positive
- **threshold_delta**: step length used to determine convergence, must be greater than or equal to 4.4e-16
- **contraction_factor**: amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1

Merit Functions

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions that can be specified with the `merit_function` control. The options are as follows:

- **merit_max**: based on $\ell_\infty$ norm
- **merit_max_smooth**: based on smoothed $\ell_\infty$ norm
- **merit1**: based on $\ell_1$ norm
- **merit1_smooth**: based on smoothed $\ell_1$ norm
- **merit2**: based on $\ell_2$ norm
- **merit2_smooth**: based on smoothed $\ell_2$ norm
- **merit2_squared**: based on $\ell_2^2$ norm

The user can also specify the following to affect the merit functions:
Method Independent Controls

The only method independent controls that are currently mapped to APPS are:

- max_function_evaluations
- constraint_tolerance
- output

Note that while APPS treats the constraint tolerance separately for linear and nonlinear constraints, we apply the same value to both if the user specifies constraint_tolerance.

The APPS internal display level is mapped to the Dakota output settings as follows:

- debug: display final solution, all input parameters, variable and constraint info, trial points, search directions, and execution details
- verbose: display final solution, all input parameters, variable and constraint info, and trial points
- normal: display final solution, all input parameters, variable and constraint summaries, and new best points
- quiet: display final solution and all input parameters
- silent: display final solution

initial_delta

- Keywords Area
- method
- asynch_pattern_search
- initial_delta

Initial step size for non-gradient based optimizers

Specification

Alias: none
Argument(s): REAL
Default: 1.0

Description

If initial_delta is supplied by the user, it will be applied in an absolute sense in all coordinate directions. APPS documentation advocates choosing initial_delta to be the approximate distance from the initial point to the solution. If this is unknown, it is advisable to err on the side of choosing an initial_delta that is too large or to not specify it. In the latter case, APPS will take a full step to the boundary in each direction. Relative application of initial_delta is not available unless the user scales the problem accordingly.
6.2. METHOD

**contraction_factor**

- Keywords Area
- method
- asynch_pattern_search
- contraction_factor

Amount by which step length is rescaled

**Specification**

Alias: none
Argument(s): REAL
Default: 0.5

**Description**

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction_factor}$

**threshold_delta**

- Keywords Area
- method
- asynch_pattern_search
- threshold_delta

Stopping criteria based on step length or pattern size

**Specification**

Alias: none
Argument(s): REAL
Default: 0.01

**Description**

`threshold_delta` is the step length or pattern size used to determine convergence.

**solution_target**

- Keywords Area
- method
- asynch_pattern_search
- solution_target

Stopping criteria based on objective function value
**CHAPTER 6. KEYWORDS AREA**

**Specification**

**Alias:** solution_accuracy

**Argument(s):** REAL

**Default:** no target

**Description**

solution_target is a termination criterion. The algorithm will terminate when the function value falls below solution_target.

**synchronization**

- Keywords Area
- method
- asynch_pattern_search
- synchronization

Select how Dakota schedules function evaluations in a pattern search

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** nonblocking

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group synchronization (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>nonblocking</td>
<td>Evaluate points in the pattern until an improving point is found</td>
</tr>
</tbody>
</table>

**Description**

The synchronization specification can be used to specify the use of either blocking or nonblocking schedulers.

**blocking**

- Keywords Area
- method
- asynch_pattern_search
- synchronization
- blocking

Evaluate all points in a pattern
6.2. METHOD

Specification

Alias: none
  Argument(s): none

Description

In the blocking case, all points in the pattern are evaluated (in parallel), and if the best of these trial points is an improving point, then it becomes the next iterate. These runs are reproducible, assuming use of the same seed in the stochastic case.

nonblocking

- Keywords Area
- method
- asynch_pattern_search
- synchronization
- nonblocking

Evaluate points in the pattern until an improving point is found

Specification

Alias: none
  Argument(s): none

Description

In the nonblocking case, all points in the pattern may not be evaluated. The first improving point found becomes the next iterate. Since the algorithm steps will be subject to parallel timing variabilities, these runs will not generally be repeatable.

merit_function

- Keywords Area
- method
- asynch_pattern_search
- merit_function

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none
  Argument(s): none
  Default: merit2_squared
## Description

A merit function is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

### merit_max

- **Keywords Area**
- **method**
- **asynch_pattern_search**
- **merit_function**
- **merit_max**

Nonsmoothed merit function

### Specification

**Alias:** none

**Argument(s):** none

### Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

**merit_max:** based on $\ell_\infty$ norm

### merit_max_smooth

- **Keywords Area**
- **method**
6.2. METHOD

- asynch_pattern_search
- merit_function
- merit_max_smooth

Smoothed merit function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

- **merit_max_smooth:** based on smoothed $\ell_\infty$ norm

**merit1**

- Keywords Area
- method
- asynch_pattern_search
- merit_function
- merit1

Nonsmoothed merit function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

- **merit1:** based on $\ell_1$ norm

**merit1_smooth**

- Keywords Area
- method
- asynch_pattern_search
- merit_function
- merit1_smooth

Smoothed merit function
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

merit1_smooth: based on smoothed $\ell_1$ norm

merit2
  - Keywords Area
  - method
  - asynch_pattern_search
  - merit_function
  - merit2

Nonsmoothed merit function

Specification

Alias: none
  Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

merit2: based on $\ell_2$ norm

merit2_smooth
  - Keywords Area
  - method
  - asynch_pattern_search
  - merit_function
  - merit2_smooth

Smoothed merit function

Specification

Alias: none
  Argument(s): none
6.2. METHOD

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

\textit{merit2\_smooth}: based on smoothed $\ell_2$ norm

\textbf{merit2\_squared}

- Keywords Area
- method
- asynch\_pattern\_search
- merit\_function
- merit2\_squared

Nonsmoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

\textit{merit2\_squared}: based on $\ell_2^2$ norm

\textbf{constraint\_penalty}

- Keywords Area
- method
- asynch\_pattern\_search
- constraint\_penalty

Multiplier for the penalty function

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds \texttt{constraint\_penalty} times the sum of squares of the constraint violations to the objective function. The default value of \texttt{constraint\_penalty} is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.
smoothing_factor

- Keywords Area
- method
- asynch_pattern_search
- smoothing_factor

Smoothing value for smoothed penalty functions

**Specification**

Alias: none

- **Argument(s):** REAL
- **Default:** 0.0

**Description**

- `smoothing_parameter`: initial smoothing value for smoothed penalty functions, must be between 0 and 1 (inclusive)

**linear_inequality_constraint_matrix**

- Keywords Area
- method
- asynch_pattern_search
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

- **Argument(s):** REALLIST
- **Default:** no linear inequality constraints
6.2. **METHOD**

**Description**

In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by `linear_inequality_lower_bounds`, and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

**linear_inequality_lower_bounds**

- **Keywords Area**
- **method**
- **asynch_pattern_search**
- **linear_inequality_lower_bounds**

Define lower bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- **linear_constraints**

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = -infinity

**Description**

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see `objective_functions`), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$A x \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than `+bigRealBoundSize` (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than `-bigRealBoundSize` are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since `-DBL_MAX < -bigRealBoundSize`).
linear_inequality_upper_bounds

- Keywords Area
- method
- asynch_pattern_search
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

**Argument(s):** REALLIST

**Default:** vector values = 0.

**Description**

In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$ a_l \leq Ax \leq a_u $$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$ Ax \leq 0.0 $$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.e+30$, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

- Keywords Area
- method
- asynch_pattern_search
- linear_inequality_scale_types

Specify how each linear inequality constraint is scaled
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

`linear_inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear_inequality_scales` must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x_j$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

`linear_inequality_scales`

- Keywords Area
- method
- asynch_pattern_search
- linear_inequality_scales

Define the characteristic values to scale linear inequalities
CHAPTER 6. KEYWORDS AREA

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

Description

Each entry in linear_inequality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of linear_inequality_scale_type:

- scale_type - behavior of linear_inequality_scales
- 'none' - ignored
- 'value' - required
- 'auto' - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear_equality_constraint_matrix

- Keywords Area
- method
- asynch_pattern_search
- linear_equality_constraint_matrix

Define coefficients of the linear equalities
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: no linear equality constraints

Description
In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

linear_equality_targets

- Keywords Area
- method
- asynch_pattern_search
- linear_equality_targets

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0.0$$
linear\_equality\_scale\_types

- Keywords Area
- method
- asynch\_pattern\_search
- linear\_equality\_scale\_types

Specify how each linear equality constraint is scaled

**Topics**
This keyword is related to the topics:

- linear\_constraints

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** vector values = "none"

**Description**

linear\_equality\_scale\_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear\_equality\_scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$
$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$
$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$
$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.
6.2. METHOD

linear.equality.scales

- Keywords Area
- method
- asynch_pattern_search
- linear.equality.scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none
Argument(s): REALLIST
Default: vector values = 1 (no scaling)

Description

Each entry in linear.equality.scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$\tilde{a}_L = a_L + A_i \text{diag}(x_M) \tilde{x} + x_O \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq A_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$. 
constraint_tolerance

- Keywords Area
- method
- asynch_pattern_search
- constraint_tolerance

The maximum allowable value of constraint violation still considered to be feasible.

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
Argument(s): REAL
Default: Library default

Description

The constraint_tolerance specification determines the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint function is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

This specification is currently meaningful for the NPSOL, NLSSOL, DOT and CONMIN constrained optimizers.

Defaults

Defaults can vary depending on the method.

- DOT constrained optimizers: 0.003
- NPSOL: dependent upon the machine precision, typically on the order of 1.e-8 for double precision computations

max_function_evaluations

- Keywords Area
- method
- asynch_pattern_search
- max_function_evaluations

Stopping criteria based on number of function evaluations
6.2. METHOD

Topics
This keyword is related to the topics:
- method\_independent\_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max\_iterations.

scaling
- Keywords Area
- method
- asynch\_pattern\_search
- scaling
  Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:
- method\_independent\_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale\_types and *\_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.
- a `*scales` keyword, which gives characteristic values

- a `*scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. *none* (default) - no scaling, value of `*scales` keyword is ignored

2. *value* - multiplicative scaling

3. *auto* - automatic scaling

   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

   - two-sided bounds scaled into the interval [0,1];
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. *log* - logarithmic scaling

   First, any characteristic values from the optional `*scales` specification are applied. Then logarithm base 10 scaling is applied.

   Logarithmic scaling is not available for linear constraints.

   When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_type`, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- *none*, *auto*, *log* - optional
- *value* - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than `1.0e10*DBL_MIN`. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
Examples
The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

**model_pointer**

- **Keywords Area**
- **method**
- **asynch_pattern_search**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**
This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**
The **model_pointer** is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

**Usage Tips**
When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See **block_pointer** for details about pointers.
**Examples**

```ini
environment
    tabular_graphics_data
    method_pointer = 'UQ'

method
    id_method = 'UQ'
    model_pointer = 'SURR'
    sampling,
        samples = 10
        seed = 98765 rng rnum2
        response_levels = 0.1 0.2 0.6
            0.1 0.2 0.6
            0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
    id_model = 'SURR'
        surrogate global,
        dace_method_pointer = 'DACE'
        polynomial quadratic

method
    id_method = 'DACE'
    model_pointer = 'DACE_M'
    sampling sample_type lhs
        samples = 121 seed = 5034 rng rnum2

model
    id_model = 'DACE_M'
        single
        interface_pointer = 'I1'

variables
    uniform_uncertain = 2
        lower_bounds = 0. 0.
        upper_bounds = 1. 1.
        descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
        system asynch evaluation_concurrency = 5
        analysis_driver = 'text_book'

responses
    response_functions = 3
    no_gradients
    no_hessians
```

6.2.31 **mesh_adaptive_search**

- **Keywords Area**
- **method**
- **mesh_adaptive_search**

Finds optimal variable values using adaptive mesh-based search
6.2. METHOD

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>function_precision</td>
<td>Specify the maximum precision of the analysis code responses</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>history_file</td>
<td>Name of file where mesh adaptive search records all evaluation points.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>display_format</td>
<td>Information to be reported from mesh adaptive search’s internal records.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>variable_neighborhood_search</td>
<td>Percentage of evaluations to do to escape local minima.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>neighbor_order</td>
<td>Number of dimensions in which to perturb categorical variables.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>display_all_evaluations</td>
<td>Shows mesh adaptive search’s internally held list of all evaluations</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

### Description

The mesh adaptive direct search algorithm[9] is a derivative-free generalized pattern search in which the set of points evaluated becomes increasingly dense, leading to good convergence properties. It can handle unconstrained problems as well as those with bound constraints and general nonlinear constraints. Furthermore, it can handle continuous, discrete, and categorical variables.

### Default Behavior

By default, mesh_adaptive_search operates on design variables. The types of variables can be expanded through the use of the active keyword in the variables block in the Dakota input file. Categorical variables, however, must be limited to design variables.

### Expected Outputs

The best objective function value achieved and associated parameter and constraint values can be found at the end of the Dakota output. The method’s internally summarized iteration history will appear in the screen output. It also generates a history file containing a list of all function evaluations done.

### Additional Discussion

The mesh adaptive direct search method is made available in Dakota through the NOMAD software[2], available to the public under the GNU LGPL from [http://www.gerad.ca/nomad](http://www.gerad.ca/nomad).

### Examples

The following is an example of a Dakota input file that makes use of mesh_adaptive_search to optimize the textbook function.

```
method,
    mesh_adaptive_search
    seed = 1234

variables,
    continuous_design = 3
    initial_point = -1.0 1.5 2.0
    upper_bounds = 10.0 10.0 10.0
    lower_bounds = -10.0 -10.0 -10.0
    descriptors = 'x1' 'x2' 'x3'

interface,
    direct
```
6.2. METHOD

```
analysis_driver = 'text_book'

responses,
  objective_functions = 1
  no_gradients
  no_hessians

The best function value and associated parameters are found at the end of the Dakota output.

<<<< Function evaluation summary: 674 total (674 new, 0 duplicate)
<<<< Best parameters
  1.0000000000e+00 x1
  1.0000000000e+00 x2
  1.0000000000e+00 x3
<<<< Best objective function = 1.0735377280e-52
<<<< Best data captured at function evaluation 658

A NOMAD-generated iteration summary is also printed to the screen.

MADS run {
  BBE OBJ
    1  17.0625000000
    2  1.0625000000
    13  0.0625000000
    24  0.0002441406
    41  0.0000314713
    43  0.0000028610
    54  0.000000037
    83  0.000000000
   105  0.000000000
   112  0.000000000
   114  0.000000000
   135  0.000000000
   142  0.000000000
   153  0.000000000
   159  0.000000000
   171  0.000000000
   193  0.000000000
   200  0.000000000
   207  0.000000000
   223  0.000000000
   229  0.000000000
   250  0.000000000
   266  0.000000000
   282  0.000000000
   288  0.000000000
   314  0.000000000
   320  0.000000000
   321  0.000000000
   327  0.000000000
   354  0.000000000
   361  0.000000000
   372  0.000000000
   373  0.000000000
   389  0.000000000
   400  0.000000000
   417  0.000000000
   444  0.000000000
   459  0.000000000
```
blackbox evaluations : 674
best feasible solution : ( 1 1 1 ) h=0 f=1.073537728e-52

**function_precision**

- Keywords Area
- method
- mesh_adaptive_search
- function_precision

Specify the maximum precision of the analysis code responses

**Specification**

*Alias*: none

*Argument(s)*: REAL

*Default*: 1.0e-10

**Description**

The `function_precision` control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.
6.2. METHOD

• seed

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```plaintext
method sampling
  sample_type lhs
  samples = 10
  seed = 15347
```

history_file

• Keywords Area

• method

• mesh_adaptive_search

• history_file

Name of file where mesh adaptive search records all evaluation points.

Specification

Alias: none

Argument(s): STRING

Default: mads_history
CHAPTER 6. KEYWORDS AREA

Description
The history_file is used to specify the name of a file to which mesh adaptive direct search will write its own list of evaluated points.

Default Behavior
By default, mesh adaptive direct search will write the list of evaluation points in a file named mads_history-xxxx, where xxxx corresponds to a randomly generated number.

Examples
The example below shows the syntax for specifying the name of the history file.

```python
method
  mesh_adaptive_search
  history_file = 'output.log'
  seed = 1234
```

display_format

- Keywords Area
- method
- mesh_adaptive_search
- display_format

Information to be reported from mesh adaptive search’s internal records.

Specification
Alias: none
Argument(s): STRING

Description
The display_format keyword is used to specify the set of information to be reported by the mesh adaptive direct search method. This is information mostly internal to the method and not reported via Dakota output.

Default Behavior
By default, only the number of function evaluations (bbe) and the objective function value (obj) are reported. The full list of options is as follows. Note that case does not matter.

- BBE: Blackbox evaluations.
- BBO: Blackbox outputs.
- EVAL: Evaluations (includes cache hits).
- MESH_INDEX: Mesh index.
- MESH_SIZE: Mesh size parameter.
- OBJ: Objective function value.
- POLL_SIZE: Poll size parameter.
6.2. METHOD

- SOL: Solution, with format iSOLj where i and j are two (optional) strings: i will be displayed before each coordinate, and j after each coordinate (except the last).

- STAT_AVG: The AVG statistic.

- STAT_SUM: The SUM statistic defined by argument.

- TIME: Wall-clock time.

- VARi: Value of variable i. The index 0 corresponds to the first variable.

Expected Outputs
A list of the requested information will be printed to the screen.

Usage Tips
This will most likely only be useful for power users who want to understand and/or report more detailed information on method behavior.

Examples
The following example shows the syntax for specifying display_format. Note that all desired information options should be listed within a single string.

```
method
    mesh_adaptive_search
    display_format 'bbe obj poll_size'
    seed = 1234
```

Below is the output reported for the above example.

```
MADS run {

BBE OBJ POLL_SIZE

   1    17.0625000000  2.0000000000 2.0000000000 2.0000000000
   2    1.0625000000  2.0000000000 2.0000000000 2.0000000000
  13    0.0625000000  1.0000000000 1.0000000000 1.0000000000
  24    0.0002441406  0.5000000000 0.5000000000 0.5000000000
  41    0.0000317413  0.1250000000 0.1250000000 0.1250000000
  43    0.0000286010  0.2500000000 0.2500000000 0.2500000000
  54    0.0000032371  0.1250000000 0.1250000000 0.1250000000
  83    0.0000000000  0.0078125000 0.0078125000 0.0078125000
 105    0.0000000000  0.0009765625 0.0009765625 0.0009765625
 112    0.0000000000  0.0000765625 0.0000765625 0.0000765625
 123    0.0000000000  0.0000093750 0.0000093750 0.0000093750
 142    0.0000000000  0.0000000488 0.0000000488 0.0000000488
 153    0.0000000000  0.0000000488 0.0000000488 0.0000000488
 159    0.0000000000  0.0000000488 0.0000000488 0.0000000488
 171    0.0000000000  0.0000000488 0.0000000488 0.0000000488
 193    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 200    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 207    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 223    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 229    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 250    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 266    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 282    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 288    0.0000000000  0.0000000610 0.0000000610 0.0000000610
 314    0.0000000000  0.0000000610 0.0000000610 0.0000000610
```

671
blackbox evaluations : 674
best feasible solution : ( 1 1 1 ) h=0 f=1.073537728e-52

See Also

These keywords may also be of interest:

- display_all_evaluations

variable_neighborhood_search

- Keywords Area
- method
- mesh_adaptive_search
- variable_neighborhood_search

Percentage of evaluations to do to escape local minima.

Specification

Alias: none

Argument(s): REAL

Default: 0.0
6.2. **METHOD**

**Description**

The `variable_neighborhood_search` keyword is used to set the percentage (in decimal form) of function evaluations used to escape local minima. The mesh adaptive direct search method will try to perform a maximum of that percentage of the function evaluations within this more extensive search.

**Default Behavior**

By default, `variable_neighborhood_search` is not used.

**Usage Tips**

Using `variable_neighborhood_search` results in an increased number of function evaluations. If the desired result is a local minimum, the added cost is of little or no value, so the recommendation is not to use it. If the desired result is the best local minimum possible within a computational budget, then there is value in setting this parameter. Note that the higher the value, the greater the computational cost.

**Examples**

The following example shows the syntax used to set `variable_neighborhood_search`.

```
method
  mesh_adaptive_search
    seed = 1234
    variable_neighborhood_search = 0.1
```

**neighbor_order**

- **Keywords Area**
- **method**
- **mesh_adaptive_search**
- **neighbor_order**

Number of dimensions in which to perturb categorical variables.

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

The `neighbor_order` keyword allows the user to specify the number of categorical dimensions to perturb when determining neighboring points that will be used by the mesh adaptive direct search method to augment its search. When greater than 1, the neighbors are defined from the tensor product of the admissible 1-dimensional perturbations.

**Default Behavior**

By default, the categorical neighbors will be defined by perturbing only one categorical variable at a time (according to the corresponding adjacency_matrix; see `adjacency_matrix`) while leaving the others fixed at their current values.

**Usage Tips**

The maximum meaningful value `neighbor_order` can take on is the number of categorical variables.
Examples

In this example, suppose we have the following categorical variables and associated adjacency matrices.

```plaintext
variables
discrete_design_set
  real = 2
categorical yes yes
num_set_values = 3 5
set_values = 1.2 2.3 3.4
1.2 3.3 4.4 5.5 7.7
adjacency_matrix = 1 1 0
  1 1 1
  0 1 1
  1 0 1 0 1
  0 1 0 1 0
  1 0 1 0 1
```

Also suppose that we have the following method specification.

```plaintext
method
mesh_adaptive_search
  seed = 1234
```

If the mesh adaptive direct search is at the point (1.2, 1.2), then the neighbors will be defined by the default 1-dimensional perturbations and would be the following:

```plaintext
(2.3, 1.2)
(1.2, 4.4)
(1.2, 7.7)
```

If, instead, the method specification is the following:

```plaintext
method
mesh_adaptive_search
  seed = 1234
  neighbor_order = 2
```

The neighbors will be defined by 2-dimensional perturbations defined from the tensor product of the 1-dimensional perturbation and would be the following:

```plaintext
(2.3, 1.2)
(2.3, 4.4)
(2.3, 7.7)
(1.2, 4.4)
(1.2, 7.7)
```

See Also

These keywords may also be of interest:

- adjacency_matrix
6.2. METHOD

display_all_evaluations
- Keywords Area
- method
- mesh_adaptive_search
- display_all_evaluations

Shows mesh adaptive search’s internally held list of all evaluations

Specification

Alias: none
Argument(s): none
Default: false

Description

If set, display_all_evaluations will instruct the mesh adaptive direct search method to print out its own record of all evaluations. The information reported may be controlled using display_format.

Default Behavior
By default, mesh adaptive direct search does not report information on all evaluations, only on those for which an improvement in the objective function is found.

Expected Outputs
The information specified by display_format will be reported to the screen for every function evaluation.

Usage Tips
This will most likely only be useful for power users who want to understand and/or report more detailed information on method behavior.

Examples

The following example shows the syntax for specifying display_all_evaluations.

```
method
  mesh_adaptive_search
  display_all_evaluations
  max_function_evaluations=20
  seed = 1234

Note that the output below reports information (default for display_format) for all function evaluations.

MADS run {

  BBE OBJ

  1 17.0625000000
  2 1.0625000000
  3 1297.0625000000
  4 257.0625000000
  5 81.0625000000
  6 151.0625000000
  7 1051.0625000000
  8 40.0625000000
  9 17.0625000000
```
That is in contrast with what would be reported by default.

See Also

These keywords may also be of interest:

- display_format

max_iterations

- Keywords Area
- method
- mesh_adaptive_search
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls
6.2. **METHOD**

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also `max_function_evaluations`.

`max_function_evaluations`

- Keywords Area
  - method
  - mesh_adaptive_search
  - `max_function_evaluations`

  Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- `method_independent_controls`

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 1000

**Description**

The maximum number of function evaluations (default: 1000). See also `max_iterations`.

`scaling`

- Keywords Area
  - method
  - mesh_adaptive_search
  - `scaling`

  Turn on scaling for variables, responses, and constraints
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and _scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of _scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales
The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples
The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

model_pointer

- Keywords Area
- method
- mesh_adaptive_search
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations
needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model
block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-
pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                     0.1 0.2 0.6
                     0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'
6.2. METHOD

interface
id_interface = 'I1'
system async evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.32 moga

- **Keywords Area**
- method
- moga

Multi-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)

**Topics**
This keyword is related to the topics:

- package.jega

**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>fitness_type</td>
<td>Select the fitness type for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>replacement_type</td>
<td>Select a replacement type for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>niching_type</td>
<td>Specify the type of niching pressure for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_type</td>
<td>Select the convergence type for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>postprocessor_type</td>
<td>Post process the final solution from moga</td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>-----------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-constraint_matrix</td>
<td>Define coefficients of the linear inequality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-lower_bounds</td>
<td>Define lower bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-upper_bounds</td>
<td>Define upper bounds for the linear inequality constraint</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scale_types</td>
<td>Specify how each linear inequality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_inequality_-scales</td>
<td>Define the characteristic values to scale linear inequalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-constraint_matrix</td>
<td>Define coefficients of the linear equalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-targets</td>
<td>Define target values for the linear equality constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-scale_types</td>
<td>Specify how each linear equality constraint is scaled</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>linear_equation_-scales</td>
<td>Define the characteristic values to scale linear equalities</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_-evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
</tbody>
</table>
### Description

moga stands for Multi-objective Genetic Algorithm, which is a global optimization method that does Pareto optimization for multiple objectives. It supports general constraints and a mixture of real and discrete variables.

**Constraints**

moga can utilize linear constraints using the keywords:  
- linear_inequality_constraint_matrix  
- linear_inequality_lower_bounds  
- linear_inequality_upper_bounds  
- linear_inequality_scale_types  
- linear_inequality_scales  
- linear_equality_constraint_matrix  
- linear_equality_targets  
- linear_equality_scale_types  
- linear_equality_scales

**Configuration**

The genetic algorithm configurations are:

1. **fitness**
2. **replacement**

### Table: moga Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>population_size</td>
<td>Set the initial population size in JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td>log_file</td>
<td>Specify the name of a log file</td>
</tr>
<tr>
<td>Optional</td>
<td>print_each_pop</td>
<td>Print every population to a population file</td>
</tr>
<tr>
<td>Optional</td>
<td>initialization_type</td>
<td>Specify how to initialize the population</td>
</tr>
<tr>
<td>Optional</td>
<td>crossover_type</td>
<td>Select a crossover type for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td>mutation_type</td>
<td>Select a mutation type for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>
3. niching
4. convergence
5. postprocessor
6. initialization
7. crossover
8. mutation
9. population size

The steps followed by the algorithm are listed below. The configurations will effect how the algorithm completes each step.

**Stopping Criteria**
The *moga* method respects the `max_iterations` and `max_function_evaluations` method independent controls to provide integer limits for the maximum number of generations and function evaluations, respectively.

The algorithm also stops when convergence is reached. This involves repeated assessment of the algorithm’s progress in solving the problem, until some criterion is met.

The specification for convergence in a *moga* can either be `metric_tracker` or can be omitted all together. If omitted, no convergence algorithm will be used and the algorithm will rely on stopping criteria only.

**Outputs**
The *moga* method respects the `output` method independent control to vary the amount of information presented to the user during execution.

The final results are written to the Dakota tabular output. Additional information is also available - see the `log_file` and `print_each_pop` keywords.

Note that *moga* and SOGA create additional output files during execution. “finaldata.dat” is a file that holds the final set of Pareto optimal solutions after any post-processing is complete. “discards.dat” holds solutions that were discarded from the population during the course of evolution.

It can often be useful to plot objective function values from these files to visually see the Pareto front and ensure that finaldata.dat solutions dominate discards.dat solutions. The solutions are written to these output files in the format "Input1...InputN..Output1...OutputM".

**Important Notes**
The pool of potential members is the current population and the current set of offspring.

Choice of fitness assessors is strongly related to the type of replacement algorithm being used and can have a profound effect on the solutions selected for the next generation.

If using the fitness types `layer_rank` or `domination_count`, it is strongly recommended that you use the `replacement_type below_limit` (although the roulette wheel selectors can also be used).

The functionality of the domination_count selector of JEGA v1.0 can now be achieved using the domination_count fitness type and below_limit replacement type.

**Theory**
The basic steps of the *moga* algorithm are as follows:

1. Initialize the population
2. Evaluate the population (calculate the values of the objective function and constraints for each population member)
3. Loop until converged, or stopping criteria reached
   (a) Perform crossover
   (b) Perform mutation
   (c) Evaluate the new population
   (d) Assess the fitness of each member in the population
   (e) Replace the population with members selected to continue in the next generation
   (f) Apply niche pressure to the population
   (g) Test for convergence

4. Perform post processing

If moga is used in a hybrid optimization method (which requires one optimal solution from each individual optimization method to be passed to the subsequent optimization method as its starting point), the solution in the Pareto set closest to the "utopia" point is given as the best solution. This solution is also reported in the Dakota output.

This "best" solution in the Pareto set has minimum distance from the utopia point. The utopia point is defined as the point of extreme (best) values for each objective function. For example, if the Pareto front is bounded by (1,100) and (90,2), then (1,2) is the utopia point. There will be a point in the Pareto set that has minimum L2-norm distance to this point, for example (10,10) may be such a point.

If moga is used in a method which may require passing multiple solutions to the next level (such as the surrogate_based_global method or hybrid methods), the orthogonal_distance postprocessor type may be used to specify the distances between each solution value to winnow down the solutions in the full Pareto front to a subset which will be passed to the next iteration.

See Also
These keywords may also be of interest:

- soga

fitness_type

- Keywords Area
- method
- moga
- fitness_type

Select the fitness type for JEGA methods

Specification

Alias: none
Argument(s): none
Default: domination_count
### Keywords Area

**Required/-Optional**

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td>layer_rank</td>
<td>Assign each member to a layer, based on domination the rank based on layers</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>domination_count</td>
<td>Rank each member by the number of members that dominate it</td>
</tr>
</tbody>
</table>

**Description**

The two JEGA methods use different fitness types, which are described on their respective pages.

**layer_rank**

- **Keywords Area**
- **method**
- **moga**
- **fitness_type**
- **layer_rank**

Assign each member to a layer, based on domination the rank based on layers

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The **fitness_type: layer_rank** has been specifically designed to avoid problems with aggregating and scaling objective function values and transforming them into a single objective.

The **layer_rank** fitness assessor works by assigning all non-dominated designs a layer of 0, then from what remains, assigning all the non-dominated a layer of 1, and so on until all designs have been assigned a layer. The values are negated to follow the higher-is-better fitness convention.

Use of the **below_limit** selector with the **layer_rank** fitness assessor has the effect of keeping all those designs whose layer is below a certain threshold again subject to the shrinkage limit.

**domination_count**

- **Keywords Area**
- **method**
- **moga**
6.2. **METHOD**

- **fitness_type**
- **domination_count**

Rank each member by the number of members that dominate it

### Specification

**Alias:** none  
**Argument(s):** none

### Description

The **fitness_type: domination_count** has been specifically designed to avoid problems with aggregating and scaling objective function values and transforming them into a single objective.

Instead, the **domination_count** fitness assessor works by ordering population members by the negative of the number of designs that dominate them. The values are negated in keeping with the convention that higher fitness is better.

The **layer_rank** fitness assessor works by assigning all non-dominated designs a layer of 0, then from what remains, assigning all the non-dominated a layer of 1, and so on until all designs have been assigned a layer. Again, the values are negated for the higher-is-better fitness convention.

Use of the **below_limit** selector with the **domination_count** fitness assessor has the effect of keeping all designs that are dominated by fewer than a limiting number of other designs subject to the shrinkage limit.

Using it with the **layer_rank** fitness assessor has the effect of keeping all those designs whose layer is below a certain threshold again subject to the shrinkage limit.

### replacement_type

- **Keywords Area**
- **method**
- **moga**
- **replacement_type**

Select a replacement type for JEGA methods

### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** below_limit

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>elitist</td>
<td>Use the best designs to form a new population</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>roulette_wheel</th>
<th>Replace population</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique_roulette_wheel</td>
<td>Replace population</td>
</tr>
<tr>
<td>below_limit</td>
<td>Limit number of designs dominating those kept</td>
</tr>
</tbody>
</table>

**Description**

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs.

In `roulette_wheel` replacement, each design is conceptually allotted a portion of a wheel proportional to its fitness relative to the fitnesses of the other Designs. Then, portions of the wheel are chosen at random and the design occupying those portions are duplicated into the next population. Those Designs allotted larger portions of the wheel are more likely to be selected (potentially many times). `unique_roulette_wheel` replacement is the same as `roulette_wheel` replacement, with the exception that a design may only be selected once. The `below_limit` selector attempts to keep all designs for which the negated fitness is below a certain limit. The values are negated to keep with the convention that higher fitness is better. The inputs to the `below_limit` selector are the limit as a real value, and a `shrinkage_percentage` as a real value. The `shrinkage_percentage` defines the minimum amount of selections that will take place if enough designs are available. It is interpreted as a percentage of the population size that must go on to the subsequent generation. To enforce this, `below_limit` makes all the selections it would make anyway and if that is not enough, it takes the remaining that it needs from the best of what is left (effectively raising its limit as far as it must to get the minimum number of selections). It continues until it has made enough selections. The `shrinkage_percentage` is designed to prevent extreme decreases in the population size at any given generation, and thus prevent a big loss of genetic diversity in a very short time. Without a shrinkage limit, a small group of “super” designs may appear and quickly cull the population down to a size on the order of the limiting value. In this case, all the diversity of the population is lost and it is expensive to re-diversify and spread the population. The `replacement_type` for a SOGA may be `roulette_wheel`, `unique_roulette_wheel`, `elitist`, or `favor_feasible`. The `elitist` selector simply chooses the required number of designs taking the most fit. For example, if 100 selections are requested, then the top 100 designs as ranked by fitness will be selected and the remaining will be discarded. The `favor_feasible` replacement type first considers feasibility as a selection criteria. If that does not produce a “winner” then it moves on to considering fitness value. Because of this, any fitness assessor used with the `favor_feasible` selector must only account objectives in the creation of fitness. Therefore, there is such a fitness assessor and it’s use is enforced when the `favor_feasible` selector is chosen. In that case, and if the output level is set high enough, a message will be presented indicating that the `weighted_sum.only` fitness assessor will be used.

**elitist**

- Keywords Area
- method
- moga
- replacement_type
6.2. METHOD

- elitist

Use the best designs to form a new population

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The *elitist* (default) setting creates a new population using (a) the *replacement_size* best individuals from the current population, (b) and *population_size* - *replacement_size* individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default *new_solutions_generated* value is set such that the entire set of newly generated individuals will be selected for replacement.

**roulette_wheel**

- Keywords Area
  - method
  - moga
  - replacement_type
  - roulette_wheel

Replace population

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The *replacement_type* of *roulette_wheel* or *unique_roulette_wheel* may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the *layer_rank* and *domination_count*, the recommended selector is the *below_limit* selector. The *below_limit* replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The *replacement_type* of *favor_feasible* is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).
unique_roulette_wheel

- Keywords Area
- method
- moga
- replacement_type
- unique_roulette_wheel

Replace population

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The `replacement_type` of `favor_feasible` is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).

**below_limit**

- Keywords Area
- method
- moga
- replacement_type
- below_limit

Limit number of designs dominating those kept

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 6
Description
The **below_limit** replacement will only keep designs that are dominated by fewer than a limiting number of other designs.

- **shrinkage_fraction**
  - **Keywords Area**
  - method
  - moga
  - replacement_type
  - below_limit
  - shrinkage_fraction

Decrease the population size by a percentage

Specification
**Alias:** shrinkage_percentage
**Argument(s):** REAL
**Default:** 0.9

Description
As of JEGA v2.0, all replacement types are common to both MOGA and SOGA. They include the **roulette_wheel**, **unique_roulette_wheel_elitist**, and **below_limit** selectors. In roulette_wheel replacement, each design is conceptually allotted a portion of a wheel proportional to its fitness relative to the fitnesses of the other Designs. Then, portions of the wheel are chosen at random and the design occupying those portions are duplicated into the next population. Those Designs allotted larger portions of the wheel are more likely to be selected (potentially many times). **unique_roulette_wheel replacement is the same as roulette_wheel replacement**, with the exception that a design may only be selected once. The **below_limit** selector attempts to keep all designs for which the negated fitness is below a certain limit. The inputs to the **below_limit** selector are the limit as a real value, and a **shrinkage_percentage** as a real value. The **shrinkage_percentage** defines the minimum amount of selections that will take place if enough designs are available. It is interpreted as a percentage of the population size that must go on to the subsequent generation. To enforce this, **below_limit** makes all the selections it would make anyway and if that is not enough, it takes the remaining that it needs from the best of what is left (effectively raising its limit as far as it must to get the minimum number of selections). It continues until it has made enough selections. The **shrinkage_percentage** is designed to prevent extreme decreases in the population size at any given generation, and thus prevent a big loss of genetic diversity in a very short time. Without a shrinkage limit, a small group of "super" designs may appear and quickly cull the population down to
a size on the order of the limiting value. In this case, all the diversity of the population is lost and it is expensive to re-diversify and spread the population. The elitist selector simply chooses the required number of designs taking the most fit. For example, if 100 selections are requested, then the top 100 designs as ranked by fitness will be selected and the remaining will be discarded.

**niching_type**

- **Keywords Area**
- **method**
- **moga**
- **niching_type**

Specify the type of niching pressure

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** No niche pressure

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> <em>(Choose One)</em></td>
<td><strong>Group 1</strong></td>
<td><strong>radial</strong></td>
<td>Set niching distance to percentage of non-dominated range</td>
</tr>
</tbody>
</table>

**distance** Enforce minimum Euclidean distance between designs

**max_designs** Limit number of solutions to remain in the population

**Description**

The purpose of niching is to encourage differentiation along the Pareto frontier and thus a more even and uniform sampling. This is typically accomplished by discouraging clustering of design points in the performance space. In JE-GA, the application of niche pressure occurs as a secondary selection operation. The nicher is given a chance to perform a pre-selection operation prior to the operation of the selection (replacement) operator, and is then called to perform niching on the set of designs that were selected by the selection operator.

The radial nicher takes information input from the user to compute a minimum allowable distance between designs in the performance space and acts as a secondary selection operator whereby it enforces this minimum distance. The distance nicher requires that solutions must be separated from other solutions by a minimum distance in each dimension (vs. Euclidean distance for the radial niching). After niching is complete, all designs in the population will be at least the minimum distance from one another in all directions.

The radial niche pressure applicator works by enforcing a minimum Euclidean distance between designs in the performance space at each generation. The algorithm proceeds by starting at the (or one of the) extreme
designs along objective dimension 0 and marching through the population removing all designs that are too close to the current design. One exception to the rule is that the algorithm will never remove an extreme design which is defined as a design that is maximal or minimal in all but 1 objective dimension (for a classical 2 objective problem, the extreme designs are those at the tips of the non-dominated frontier). The distance nicher enforces a minimum distance in each dimension.

The designs that are removed by the nicher are not discarded. They are buffered and re-inserted into the population during the next pre-selection operation. This way, the selector is still the only operator that discards designs and the algorithm will not waste time “re-filling” gaps created by the nicher.

The radial nicher requires as input a vector of fractions with length equal to the number of objectives. The elements of the vector are interpreted as percentages of the non-dominated range for each objective defining a minimum distance to all other designs. All values should be in the range \((0, 1)\). The minimum allowable distance between any two designs in the performance space is the Euclidian (simple square-root-sum-of-squares calculation) distance defined by these percentages. The distance nicher has a similar input vector requirement, only the distance is the minimum distance in each dimension.

The max_designs niche pressure applicator is designed to choose a limited number of solutions to remain in the population. That number is specified by num_designs. It does so in order to balance the tendency for populations to grow very large and thus consuming too many computer resources. It operates by ranking designs according to their fitness standing and a computed count of how many other designs are too close to them. Too close is a function of the supplied niche_vector, which specifies the minimum distance between any two points in the performance space along each dimension individually. Once the designs are all ranked, the top \(c\) \(\leq\) num_designs designs are kept in the population and the remaining ones are buffered or discarded. Note that like other niching operators, this one will not discard an extreme design.

radial

- Keywords Area
- method
- moga
- niching_type
- radial

Set niching distance to percentage of non-dominated range

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** 0.01 for all objectives

**Description**

The radial nicher requires as input a vector of fractions with length equal to the number of objectives. The elements of the vector are interpreted as percentages of the non-dominated range for each objective defining a minimum distance to all other designs. All values should be in the range \((0, 1)\). The minimum allowable distance between any two designs in the performance space is the Euclidian (simple square-root-sum-of-squares calculation) distance defined by these percentages. The distance nicher has a similar input vector requirement, only the distance is the minimum distance in each dimension.
CHAPTER 6. KEYWORDS AREA

distance

- Keywords Area
- method
- moga
- niching_type
- distance

Enforce minimum Euclidean distance between designs

**Specification**

Alias: none

**Argument(s):** REALLIST

**Description**

Currently, the only niche pressure operators available are the radial nicher, the distance nicher, and the max_designs nicher. The radial niche pressure applicator works by enforcing a minimum Euclidean distance between designs in the performance space at each generation. The algorithm proceeds by starting at the (or one of the) extreme designs along objective dimension 0 and marching through the population removing all designs that are too close to the current design. One exception to the rule is that the algorithm will never remove an extreme design which is defined as a design that is maximal or minimal in all but 1 objective dimension (for a classical 2 objective problem, the extreme designs are those at the tips of the non-dominated frontier). The distance nicher enforces a minimum distance in each dimension.

The designs that are removed by the nicher are not discarded. They are buffered and re-inserted into the population during the next pre-selection operation. This way, the selector is still the only operator that discards designs and the algorithm will not waste time “re-filling” gaps created by the nicher.

max_designs

- Keywords Area
- method
- moga
- niching_type
- max_designs

Limit number of solutions to remain in the population

**Specification**

Alias: none

**Argument(s):** REALLIST
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_designs</td>
<td>num_designs</td>
<td>Limit the number of solutions</td>
</tr>
</tbody>
</table>

**Description**

The `max_designs` niche pressure applicator is designed to choose a limited number of solutions to remain in the population. That number is specified by `num_designs`. It does so in order to balance the tendency for populations to grow very large and thus consuming too many computer resources. It operates by ranking designs according to their fitness standing and a computed count of how many other designs are too close to them. Too close is a function of the supplied niche vector, which specifies the minimum distance between any two points in the performance space along each dimension individually. Once the designs are all ranked, the top `c\ num_designs` designs are kept in the population and the remaining ones are buffered or discarded. Note that like other niching operators, this one will not discard an extreme design.

- **num_designs**
  - Keywords Area
  - method
  - moga
  - niching_type
  - max_designs
  - num_designs

Limit the number of solutions

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** 100

**Description**

The `max_designs` niche pressure applicator is designed to choose a limited number of solutions to remain in the population. That number is specified by `num_designs`. It does so in order to balance the tendency for populations to grow very large and thus consuming too many computer resources. It operates by ranking designs according to their fitness standing and a computed count of how many other designs are too close to them. Too close is a function of the supplied niche vector, which specifies the minimum distance between any two points in the performance space along each dimension individually. Once the designs are all ranked, the top `c\ num_designs` designs are kept in the population and the remaining ones are buffered or discarded. Note that like other niching operators, this one will not discard an extreme design.
convergence_type

- Keywords Area
- method
- moga
- convergence_type

Select the convergence type for JEGA methods

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** average_fitness_tracker

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>metric_tracker</td>
<td>Track changes in the non-dominated frontier</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>percent_change</td>
<td>Define the convergence criterion for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>num_generations</td>
<td>Define the convergence criterion for JEGA methods</td>
</tr>
</tbody>
</table>

**Description**

The two JEGA methods use different convergence types, which are described on their respective pages.

All the convergence types are modified by the optional keywords `percent_change` and `num_generations`.

**metric_tracker**

- Keywords Area
- method
- moga
- convergence_type
- metric_tracker

Track changes in the non-dominated frontier
6.2. **METHOD**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** `metric_tracker`

**Description**

The moga converger (``metric_tracker``) operates by tracking various changes in the non-dominated frontier from generation to generation. When the changes occurring over a user specified number of generations fall below a user specified threshold, the algorithm stops.

If `metric_tracker` is specified, then a `percent_change` and `num_generations` must be supplied as well. These are listed as optional keywords in the input spec.

**Theory**

The `metric_tracker` converger tracks 3 metrics specific to the non-dominated frontier from generation to generation. All 3 of these metrics are computed as percent changes between the generations. In order to compute these metrics, the converger stores a duplicate of the non-dominated frontier at each generation for comparison to the non-dominated frontier of the next generation.

The first metric is one that indicates how the expanse of the frontier is changing. The expanse along a given objective is defined by the range of values existing within the non-dominated set. The expansion metric is computed by tracking the extremes of the non-dominated frontier from one generation to the next. Any movement of the extreme values is noticed and the maximum percentage movement is computed as:

\[
Em = \max \text{ over } j \text{ of } \frac{\text{abs}(\text{range}(j, \ i) - \text{range}(j, \ i-1))}{\text{range}(j, \ i-1)} \quad j=1,\text{nof}
\]

where `Em` is the max expansion metric, `j` is the objective function index, `i` is the current generation number, and `nof` is the total number of objectives. The range is the difference between the largest value along an objective and the smallest when considering only non-dominated designs.

The second metric monitors changes in the density of the non-dominated set. The density metric is computed as the number of non-dominated points divided by the hypervolume of the non-dominated region of space. Therefore, changes in the density can be caused by changes in the number of non-dominated points or by changes in size of the non-dominated space or both. The size of the non-dominated space is computed as:

\[
Vps(i) = \prod \text{ over } j \text{ of } \text{range}(j, \ i) \quad j=1,\text{nof}
\]

where `Vps(i)` is the hypervolume of the non-dominated space at generation `i` and all other terms have the same meanings as above.

The density of a given non-dominated space is then:

\[
Dps(i) = \frac{\text{Pct}(i)}{Vps(i)}
\]

where `Pct(i)` is the number of points on the non-dominated frontier at generation `i`.

The percentage increase in density of the frontier is then calculated as

\[
Cd = \frac{\text{abs}((Dps(i) - Dps(i-1))}{Dps(i-1)}
\]

where `Cd` is the change in density metric.

The final metric is one that monitors the "goodness" of the non-dominated frontier. This metric is computed by considering each design in the previous population and determining if it is dominated by any designs in the current population. All that are determined to be dominated are counted. The metric is the ratio of the number that are dominated to the total number that exist in the previous population.

As mentioned above, each of these metrics is a percentage. The tracker records the largest of these three at each generation. Once the recorded percentage is below the supplied percent change for the supplied number of generations consecutively, the algorithm is converged.
percent_change

- Keywords Area
- method
- moga
- convergence_type
- percent_change

Define the convergence criterion for JEGA methods

**Specification**

Alias: none

Argument(s): REAL

Default: 0.1

**Description**

The `percent_change` is the threshold beneath which convergence is attained whereby it is compared to the metric value computed.

num_generations

- Keywords Area
- method
- moga
- convergence_type
- num_generations

Define the convergence criterion for JEGA methods

**Specification**

Alias: none

Argument(s): INTEGER

Default: 10

**Description**

The `num_generations` is the number of generations over which the metric value should be tracked. Convergence will be attained if the recorded metric is below `percent_change` for `num_generations` consecutive generations.
6.2. **METHOD**

**postprocessor_type**

- Keywords Area
- method
- moga
- postprocessor_type

Post process the final solution from **moga**

**Specification**

**Alias**: none  
**Argument(s)**: none  
**Default**: No post-processing of solutions

<table>
<thead>
<tr>
<th>Required/ Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>orthogonal_distance</td>
<td>Get subset of Pareto front based on distance</td>
</tr>
</tbody>
</table>

**Description**

The purpose of this operation is to perform any needed data manipulations on the final solution deemed necessary. Currently the **orthogonal_distance** is the only one. It reduces the final solution set size such that a minimum distance in each direction exists between any two designs.

**orthogonal_distance**

- Keywords Area
- method
- moga
- postprocessor_type
- orthogonal_distance

Get subset of Pareto front based on distance

**Specification**

**Alias**: none  
**Argument(s)**: REALLIST  
**Default**: 0.01 for all objectives
Description

Note that MOGA and SOGA create additional output files during execution. "finaldata.dat" is a file that holds the final set of Pareto optimal solutions after any post-processing is complete. "discards.dat" holds solutions that were discarded from the population during the course of evolution. It can often be useful to plot objective function values from these files to visually see the Pareto front and ensure that finaldata.dat solutions dominate discards.dat solutions. The solutions are written to these output files in the format "Input1...InputN..Output1...OutputM". If MOGA is used in a hybrid optimization meta-iteration (which requires one optimal solution from each individual optimization method to be passed to the subsequent optimization method as its starting point), the solution in the Pareto set closest to the "utopia" point is given as the best solution. This solution is also reported in the Dakota output. This "best" solution in the Pareto set has minimum distance from the utopia point. The utopia point is defined as the point of extreme (best) values for each objective function. For example, if the Pareto front is bounded by (1,100) and (90,2), then (1,2) is the utopia point. There will be a point in the Pareto set that has minimum L2-norm distance to this point, for example (10,10) may be such a point. In SOGA, the solution that minimizes the single objective function is returned as the best solution. If moga is used in meta-iteration which may require passing multiple solutions to the next level (such as the surrogate_based_global or hybrid methods), the orthogonal_distance postprocessor type may be used to specify the distances between each solution value to winnow down the solutions in the full Pareto front to a subset which will be passed to the next iteration.

max_iterations

- Keywords Area
- method
- moga
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.
6.2. METHOD

**linear_inequality_constraint_matrix**

- **Keywords Area**
- **method**
- **moga**
- **linear_inequality_constraint_matrix**

Define coefficients of the linear inequality constraints

**Topics**

This keyword is related to the topics:

- **linear_constraints**

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** no linear inequality constraints

**Description**

In the inequality case, the constraint matrix \( A \) provides coefficients for the variables in the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where the bounds are optionally specified by \( \text{linear_inequality_lower_bounds} \), and \( \text{linear_inequality_upper_bounds} \). The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

\[
Ax \leq 0
\]

**linear_inequality_lower_bounds**

- **Keywords Area**
- **method**
- **moga**
- **linear_inequality_lower_bounds**

Define lower bounds for the linear inequality constraint

**Topics**

This keyword is related to the topics:

- **linear_constraints**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = -infinity

Description

In the inequality case, the lower \( a_l \) and upper \( a_u \) bounds provide constraint limits for the two-sided formulation:

\[
a_l \leq Ax \leq a_u
\]

Where \( A \) is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

\[
Ax \leq 0.0
\]

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).

linear_inequality_upper_bounds

- Keywords Area
- method
- moga
- linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0.
6.2. METHOD

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ (1.e+30, as defined in Minimizer) are treated as $+\infty$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\infty$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBLMAX} < -\text{bigRealBoundSize}$).

**linear_inequality_scale_types**

- **Keywords Area**
- **method**
- **moga**
- **linear_inequality_scale_types**

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

- **linear_constraints**

Specification

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** vector values = "none"

Description

**linear_inequality_scale_types** provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- **'none’** - no scaling
- **'value’** - characteristic value if this is chosen, then **linear_inequality_scales** must be specified
• ‘auto’ - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]

\[
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U
\]

\[
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O
\]

\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

**linear_inequality_scales**

- Keywords Area
- method
- moga
- linear_inequality_scales

Define the characteristic values to scale linear inequalities

**Topics**

This keyword is related to the topics:

- linear_constraints

**Specification**

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

**Description**

Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`
- ‘none’ - ignored
6.2. METHOD

- ‘value’ - required
- ‘auto’ - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U \\
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \\
\tilde{a_L} \leq \tilde{A}_i \tilde{x} \leq \tilde{a_U}
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

linear_equation_constraint_matrix

- Keywords: Area
- method
- moga

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- linear_constraints

Specification

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints

Description

In the equality case, the constraint matrix \( A \) provides coefficients for the variables on the left hand side of:

\[
Ax = a_t
\]
linear_equal	ity_targets
  • Keywords Area
  • method
  • moga
  • linear_equal
ty	ities

Define target values for the linear equality constraints

Topics
This keyword is related to the topics:
  • linear_constraints

Specification
Alias: none
  Argument(s): REALLIST
  Default: vector values = 0.

Description
In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint: $Ax = 0.0$

linear_equal	ity_scale_types
  • Keywords Area
  • method
  • moga
  • linear_equal	ity_scale_types

Specify how each linear equality constraint is scaled

Topics
This keyword is related to the topics:
  • linear_constraints
6.2. METHOD

Specification

Alias: none
Argument(s): STRINGLIST
Default: vector values = "none"

Description

linear.equality.scale_types provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- 'none' - no scaling
- 'value' - characteristic value if this is chosen, then linear.equality.scales must be specified
- 'auto' - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

linear.equality.scales

- Keywords Area
- method
- moga
- linear.equality.scales

Define the characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- linear.constraints
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): REALIST
  Default: vector values = 1 . (no scaling)

Description

Each entry in linear.equality.scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.

Scaling for linear constraints is applied after any continuous variable scaling.

For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

max.function_evaluations

- Keywords Area
- method
- moga
- max.function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method.independent.controls

Specification

Alias: none
  Argument(s): INTEGER
  Default: 1000
6.2. **METHOD**

**Description**

The maximum number of function evaluations (default: 1000). See also `max.iterations`.

**scaling**

- Keywords Area
- method
- moga
- scaling

Turn on scaling for variables, responses, and constraints

**Topics**

This keyword is related to the topics:

- `method.independent_controls`

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no scaling

**Description**

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the `method`, `variables`, and `responses` blocks. When the `scaling` keyword is omitted, all `_scale_types` and `*_scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- A `*scales` keyword, which gives characteristic values
- A `*scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. `none` (default) - no scaling, value of `*scales` keyword is ignored
2. **value** - multiplicative scaling

3. **auto** - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval \([0,1]\);
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into \([0,1]\).

4. **log** - logarithmic scaling

First, any characteristic values from the optional `*scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_type`, as described above. Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0e10*DBL_MIN\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
6.2. METHOD

population_size

- Keywords Area
- method
- moga
- population_size

Set the initial population size in JEGA methods

Specification

Alias: none
Argument(s): INTEGER
Default: 50

Description

The number of designs in the initial population is specified by the population_size. Note that the population-size only sets the size of the initial population. The population size may vary in the JEGA methods according to the type of operators chosen for a particular optimization run.

log_file

- Keywords Area
- method
- moga
- log_file

Specify the name of a log file

Specification

Alias: none
Argument(s): STRING
Default: JEGAGlobal.log

Description

New as of JEGA v2.0 is the introduction of the log_file specification. JEGA now uses a logging library to output messages and status to the user. JEGA can be configured at build time to log to both the console window and a text file, one or the other, or neither. The log_file input is a string name of a file into which to log. If the build was configured without file logging in JEGA, this input is ignored. If file logging is enabled and no log_file is specified, the default file name of JEGAGlobal.log is used.
print_each_pop

- Keywords Area
- method
- moga
- print_each_pop

Print every population to a population file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** No printing

**Description**

New to JEGA v2.0 is the introduction of the `print_each_pop` specification. It serves as a flag and if supplied, the population at each generation will be printed to a file named "population<GEN#>.dat" where <GEN#> is the number of the current generation.

initialization_type

- Keywords Area
- method
- moga
- initialization_type

Specify how to initialize the population

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** unique_random

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>simple_random</td>
<td>Create random initial solutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique_random</td>
<td>Create random initial solutions, but enforce uniqueness (default)</td>
</tr>
</tbody>
</table>
6.2. METHOD

| flat_file | Read initial solutions from file |

Description

The `initialization.type` defines how the initial population is created for the GA. There are three types:

1. `simple_random`
2. `unique_random` (default)
3. `flat_file`

Setting the size for the `flat_file` initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the `unique_random` initializer and then the `simple_random` initializer if necessary.

**simple_random**

- Keywords Area
- method
- moga
- `initialization.type`
- `simple_random`

Create random initial solutions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

`simple_random` creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs.

**unique_random**

- Keywords Area
- method
- moga
- `initialization.type`
- `unique_random`

Create random initial solutions, but enforce uniqueness (default)
CHAPTER 6. KEYWORDS AREA

**Specification**

Alias: none

  *Argument(s):* none

**Description**

*unique_random* is the same as *simple_random*, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected.

**flat_file**

- Keywords Area
- method
- moga
- initialization_type
- flat_file

Read initial solutions from file

**Specification**

Alias: none

  *Argument(s):* STRING

**Description**

*flat_file* allows the initial population to be read from a flat file. If *flat_file* is specified, a file name must be given.

Variables can be delimited in the flat file in any way you see fit with a few exceptions. The delimiter must be the same on any given line of input with the exception of leading and trailing whitespace. So a line could look like: 1.1, 2.2, 3.3 for example but could not look like: 1.1, 2.2 3.3. The delimiter can vary from line to line within the file which can be useful if data from multiple sources is pasted into the same input file. The delimiter can be any string that does not contain any of the characters .+dDeE or any of the digits 0-9. The input will be read until the end of the file. The algorithm will discard any configurations for which it was unable to retrieve at least the number of design variables. The objective and constraint entries are not required but if ALL are present, they will be recorded and the design will be tagged as evaluated so that evaluators may choose not to re-evaluate them.

Setting the size for this initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the *unique_random* initializer and then the *simple_random* initializer if necessary.

**crossover_type**

- Keywords Area
- method
- moga
- crossover_type

Select a crossover type for JEGA methods
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: shuffle_random

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>multi_point_binary</td>
<td>Use bit switching for crossover events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multi_point_parameterized_binary</td>
<td>Use bit switching to crossover each design variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multi_point_real</td>
<td>Perform crossover in real valued genome</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>shuffle_random</td>
<td>Perform crossover by choosing design variable(s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>crossover_rate</td>
<td>Specify the probability of a crossover event</td>
</tr>
</tbody>
</table>

Description

There are many crossover types available. multi_point_binary crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). multi_point_parameterized_binary crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. multi_point_real crossover performs a variable switching crossover routing at N crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by $10^6$ and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

The final crossover type is shuffle_random. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

All crossover types take a crossover_rate. The crossover rate is used to calculate the number of crossover operations that take place. The number of crossovers is equal to the rate * population_size.

**multi_point_binary**

- Keywords Area
CHAPTER 6. KEYWORDS AREA

- method
- moga
- crossover_type
- multi_point_binary

Use bit switching for crossover events

**Specification**

Alias: none

**Argument(s):** INTEGER

**Description**

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by $10^6$ and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

`multi_point_parameterized_binary`

- **Keywords Area**
- method
- moga
- crossover_type
- `multi_point_parameterized_binary`

Use bit switching to crossover each design variable

**Specification**

Alias: none

**Argument(s):** INTEGER
6.2. **METHOD**

**Description**

There are many crossover types available. `multi_point_binary` crossover requires an integer number, \( N \), of crossover points. This crossover type performs a bit switching crossover at \( N \) crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at \( N \) crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at \( N \) locations. `multi_point_real` crossover performs a variable switching crossover routing at \( N \) crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by \( 10^6 \) and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

**multi_point_real**

- **Keywords Area**
- **method**
- **moga**
- **crossover_type**
- **multi_point_real**

Perform crossover in real valued genome

**Specification**

**Alias:** none  
**Argument(s):** INTEGER

**Description**

There are many crossover types available. `multi_point_binary` crossover requires an integer number, \( N \), of crossover points. This crossover type performs a bit switching crossover at \( N \) crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at \( N \) crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at \( N \) locations. `multi_point_real` crossover performs a variable switching crossover routing at \( N \) crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by \( 10^6 \) and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.
The final crossover type is `shuffle_random`. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

### num_parents

- **Keywords Area**
- **method**
- **moga**
- **crossover_type**
- **shuffle_random**

Number of parents in random shuffle crossover
6.2. \textit{METHOD} \\

\textbf{Specification} \\
\textbf{Alias:} none \\
\textbf{Argument(s):} INTEGER \\
\textbf{Default:} 2 \\

\textbf{Description} \\
Number of parents in random shuffle crossover \\

\texttt{num\_offspring} \\
\begin{itemize}
  \item Keywords Area 
  \item method 
  \item moga 
  \item crossover\_type 
  \item shuffle\_random 
  \item num\_offspring 
\end{itemize} \\
Number of offspring in random shuffle crossover \\

\textbf{Specification} \\
\textbf{Alias:} none \\
\textbf{Argument(s):} INTEGER \\
\textbf{Default:} 2 \\

\textbf{Description} \\
Number of offspring in random shuffle crossover \\

\texttt{crossover\_rate} \\
\begin{itemize}
  \item Keywords Area 
  \item method 
  \item moga 
  \item crossover\_type 
  \item crossover\_rate 
\end{itemize} \\
Specify the probability of a crossover event \\

\textbf{Specification} \\
\textbf{Alias:} none \\
\textbf{Argument(s):} REAL \\
\textbf{Default:} 0.8
Description

The `crossover.type` controls what approach is employed for combining parent genetic information to create offspring, and the `crossover.rate` specifies the probability of a crossover operation being performed to generate a new offspring. The SCOLIB EA method supports three forms of crossover, `two_point`, `blend`, and `uniform`, which generate a new individual through combinations of two parent individuals. Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate. Uniform crossover creates offspring through random combination of coordinates from the two parents. Blend crossover generates a new individual randomly along the multidimensional vector connecting the two parents.

**mutation_type**

- Keywords Area
- method
- moga
- `mutation_type`

Select a mutation type for JEGA methods

Specification

**Alias:** none  
**Argument(s):** none  
**Default:** `replace_uniform`

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> <em>(Choose One)</em></td>
<td><strong>Group 1</strong></td>
<td><code>replace_uniform</code></td>
<td>Mutate by flipping a random bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>offset_normal</code></td>
<td>Use uniformly distributed value over range of parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>offset_cauchy</code></td>
<td>Use a Cauchy distribution for the mutation offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>offset_uniform</code></td>
<td>Set mutation offset to use a uniform distribution</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>mutation_rate</th>
<th>Set probability of a mutation</th>
</tr>
</thead>
</table>

**Description**

Five mutation types are available for selection by keyword: `replace_uniform`, `bit_random`, `offset_cauchy`, `offset_normal`, and `offset_uniform`. They are described in greater detail on their respective keyword pages.

The `offset_*` mutators all act by adding a random "offset" to a variable value. The random amount has a mean of zero in all cases. The size of the offset is controlled using the `mutation_scale` keyword, which is interpreted differently for each `offset_*` type.

The rate of mutations for all types is controlled using the `mutation_rate`. The rate is applied differently in each `mutation_type`.

*bit_random*

- **Keywords Area**
- **method**
- **moga**
- **mutation_type**
- **bit_random**

Mutate by flipping a random bit

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `bit_random` mutator introduces random variation by first converting a randomly chosen variable of a randomly chosen design into a binary string. It then flips a randomly chosen bit in the string from a 1 to a 0 or visa versa. In this mutation scheme, the resulting value has more probability of being similar to the original value.

*replace_uniform*

- **Keywords Area**
- **method**
- **moga**
- **mutation_type**
- **replace_uniform**

Use uniformly distributed value over range of parameter
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
    Argument(s): none

Description
replace_uniform introduces random variation by first randomly choosing a design variable of a randomly selected design and reassigning it to a random valid value for that variable. No consideration of the current value is given when determining the new value.

offset_normal
    • Keywords Area
    • method
    • moga
    • mutation_type
    • offset_normal

Set mutation offset to use a normal distribution

Specification
Alias: none
    Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>mutation_scale</td>
<td>Scales mutation across range of parameter</td>
</tr>
</tbody>
</table>

Description
The offset_normal mutator introduces random variation by adding a Gaussian random amount to a variable value. The random amount has a standard deviation dependent on the mutation_scale.

mutation_scale
    • Keywords Area
    • method
    • moga
    • mutation_type
    • offset_normal
    • mutation_scale

Scales mutation across range of parameter
6.2. METHOD

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** 0.15

**Description**

The mutation\_scale is a fraction in the range \([0, 1]\) and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected mutation\_type. For offset\_normal and offset\_cauchy, mutation\_scale is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For offset\_uniform, the range of possible deviation amounts is +/- 1/2 \((\text{mutation\_scale} \times \text{variable range})\).

**offset\_cauchy**

- **Keywords Area**
- method
- moga
- mutation\_type
- offset\_cauchy

Use a Cauchy distribution for the mutation offset

**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>mutation_scale</td>
<td>mutation_scale</td>
<td>Scales mutation across range of parameter</td>
</tr>
</tbody>
</table>

**Description**

The offset\_cauchy mutator introduces random variation by adding a Cauchy random amount to a variable value. The random amount has a standard deviation dependent on the mutation\_scale.

**mutation\_scale**

- **Keywords Area**
- method
- moga
- mutation\_type
- offset\_cauchy
• **mutation_scale**

Scales mutation across range of parameter

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 0.15

**Description**

The mutation scale is a fraction in the range $[0, 1]$ and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected mutation type. For offset normal and offset cauchy, mutation scale is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For offset uniform, the range of possible deviation amounts is $\pm \frac{1}{2} \times \text{mutation scale} \times \text{variable range}$.

**offset_uniform**

- Keywords Area
- method
- moga
- mutation_type
- offset_uniform

Set mutation offset to use a uniform distribution

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>mutation_scale</td>
<td>Scales mutation across range of parameter</td>
</tr>
</tbody>
</table>

**Description**

The offset_uniform mutator introduces random variation by adding a uniform random amount to a variable value. The random amount depends on the mutation scale.
6.2. METHOD

**mutation_scale**
- Keywords Area
- method
- moga
- mutation_type
- offset_uniform
- mutation_scale

Scales mutation across range of parameter

**Specification**

Alias: none
Argument(s): REAL
Default: 0.15

**Description**

The mutation_scale is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected mutation_type. For offset_normal and offset_cauchy, mutation_scale is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For offset_uniform, the range of possible deviation amounts is +/- 1/2 * (mutation_scale * variable range).

**mutation_rate**
- Keywords Area
- method
- moga
- mutation_type
- mutation_rate

Set probability of a mutation

**Specification**

Alias: none
Argument(s): REAL
Default: 0.08

**Description**

All mutation types have a mutation_rate, which controls the number of mutations performed. For replace_uniform and all the offset_* types, the number of mutations performed is the product of mutation_rate and population_size. For bit_random, it's the product of the mutation_rate, number of design variables, and population size.
seed

- **Keywords Area**
- **method**
- **moga**
- **seed**

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)

**Description**

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```plaintext
method sampling
  sample_type lhs
  samples = 10
  seed = 15347
```

**convergence_tolerance**

- **Keywords Area**
- **method**
- **moga**
- **convergence_tolerance**

Stopping criterion based on convergence of the objective function or statistics
6.2. METHOD

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

model_pointer

- Keywords Area
- method
- moga
- model_pointer

Identifier for model block to be used by a method
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.
See block_pointer for details about pointers.

Examples

environment
tagular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SRR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SRR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
6.2. METHOD

interface_pointer = ‘I1’

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = ‘x1’ ‘x2’

interface
id_interface = ‘I1’
system async evaluation_concurrency = 5
analysis_driver = ‘text_book’

responses
response_functions = 3
no_gradients
no_hessians

6.2.33 soga

- Keywords Area
- method
- soga

Single-objective Genetic Algorithm (a.k.a. Evolutionary Algorithm)

Topics
This keyword is related to the topics:

- package_jega
- global_optimization_methods

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>fitness_type</td>
<td>Description</td>
</tr>
</tbody>
</table>
| Optional            |                      | replacement_type| Select the fitness type for JEGA methods
| Optional            |                      |                | Select a replacement type for JEGA methods |
| Optional | convergence_type | Select the convergence type for JEGA methods |
| Optional | max_iterations | Stopping criterion based on number of iterations |
| Optional | linear_inequality_constraint_matrix | Define coefficients of the linear inequality constraints |
| Optional | linear_inequality_lower_bounds | Define lower bounds for the linear inequality constraint |
| Optional | linear_inequality_upper_bounds | Define upper bounds for the linear inequality constraint |
| Optional | linear_inequality_scale_types | Specify how each linear inequality constraint is scaled |
| Optional | linear_inequality_scales | Define the characteristic values to scale linear inequalities |
| Optional | linear_equality_constraint_matrix | Define coefficients of the linear equalities |
| Optional | linear_equality_targets | Define target values for the linear equality constraints |
| Optional | linear_equality_scale_types | Specify how each linear equality constraint is scaled |
| Optional | linear_equality_scales | Define the characteristic values to scale linear equalities |
| Optional | max_function_evaluations | Stopping criteria based on number of function evaluations |
### Description

`soga` stands for Single-objective Genetic Algorithm, which is a global optimization method that supports general constraints and a mixture of real and discrete variables. `soga` is part of the JEGA library.

**Constraints** `soga` can utilize linear constraints.

**Configuration**
The genetic algorithm configurations are:

1. fitness
2. replacement
3. convergence
4. initialization
5. crossover

| Optional   | scaling                  | Turn on scaling for variables, responses, and constraints
| Optional   | population_size          | Set the initial population size in JEGA methods
| Optional   | log_file                 | Specify the name of a log file
| Optional   | print_each_pop           | Print every population to a population file
| Optional   | initialization_type      | Specify how to initialize the population
| Optional   | crossover_type           | Select a crossover type for JEGA methods
| Optional   | mutation_type            | Select a mutation type for JEGA methods
| Optional   | seed                     | Seed of the random number generator
| Optional   | convergence_tolerance    | Stopping criterion based on convergence of the objective function or statistics
| Optional   | model_pointer            | Identifier for model block to be used by a method |
6. mutation
7. population size

The pool of potential members is the current population and the current set of offspring. Choice of fitness assessors is strongly related to the type of replacement algorithm being used and can have a profound effect on the solutions selected for the next generation.

Stopping Criteria
The soga method respects the `max_iterations` and `max_function_evaluations` method independent controls to provide integer limits for the maximum number of generations and function evaluations, respectively.

The algorithm also stops when convergence is reached. This involves repeated assessment of the algorithm’s progress in solving the problem, until some criterion is met.

Outputs The soga method respects the `output` method independent control to vary the amount of information presented to the user during execution.

The final results are written to the Dakota tabular output. Additional information is also available - see the `log_file` and `print_each_pop` keywords.

Theory
The basic steps of the soga algorithm are as follows:

1. Initialize the population
2. Evaluate the population (calculate the values of the objective function and constraints for each population member)
3. Loop until converged, or stopping criteria reached
   (a) Perform crossover
   (b) Perform mutation
   (c) Evaluate the new population
   (d) Assess the fitness of each member in the population
   (e) Replace the population with members selected to continue in the next generation
   (f) Test for convergence

See Also
These keywords may also be of interest:

- moga

fitness_type

- Keywords Area
- method
- soga
- fitness_type

Select the fitness type for JEGA methods
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: merit_function

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>merit_function</td>
<td>Balance goals of reducing objective function and satisfying constraints</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>constraint_penalty</td>
<td>Multiplier for the penalty function</td>
</tr>
</tbody>
</table>

Description

The two JEGA methods use different fitness types, which are described on their respective pages.

**merit_function**

- **Keywords Area**
- **method**
- **soga**
- **fitness_type**
- **merit_function**

Balance goals of reducing objective function and satisfying constraints

**Specification**

Alias: none
Argument(s): none

Description

A **merit_function** is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

**constraint_penalty**

- **Keywords Area**
- **method**
- **soga**
- **fitness_type**
- **constraint_penalty**

Multiplier for the penalty function
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): REAL
Default: 1.0

Description

The merit_function fitness assessor uses an exterior penalty function formulation to penalize infeasible designs. The specification allows the input of a constraint_penalty which is the multiplier to use on the constraint violations.

replacement_type

- Keywords Area
- method
- soga
- replacement_type

Select a replacement type for JEGA methods

Specification

Alias: none
Argument(s): none
Default: elitist

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>elitist</td>
<td>Use the best designs to form a new population</td>
</tr>
<tr>
<td></td>
<td></td>
<td>favor_feasible</td>
<td>Prioritize feasible designs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>roulette_wheel</td>
<td>Replace population</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique_roulette_wheel</td>
<td>Replace population</td>
</tr>
</tbody>
</table>

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The replacement_type of roulette_wheel or unique_roulette_wheel may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the layer_rank and domination_count, the recommended selector is the below_limit selector. The below_limit replacement will only keep designs that are dominated by fewer than a limiting number of other designs.

In roulette_wheel replacement, each design is conceptually allotted a portion of a wheel proportional to its fitness relative to the fitnesses of the other Designs. Then, portions of the wheel are chosen at random and the design occupying those portions are duplicated into the next population. Those Designs allotted larger portions of the wheel are more likely to be selected (potentially many times). unique_roulette_wheel replacement
6.2. METHOD

is the same as roulette_wheel replacement, with the exception that a design may only be selected once. The below_limit selector attempts to keep all designs for which the negated fitness is below a certain limit. The values are negated to keep with the convention that higher fitness is better. The inputs to the below_limit selector are the limit as a real value, and a shrinkage_percentage as a real value. The shrinkage_percentage defines the minimum amount of selections that will take place if enough designs are available. It is interpreted as a percentage of the population size that must go on to the subsequent generation. To enforce this, below_limit makes all the selections it would make anyway and if that is not enough, it takes the remaining that it needs from the best of what is left (effectively raising its limit as far as it must to get the minimum number of selections). It continues until it has made enough selections. The shrinkage_percentage is designed to prevent extreme decreases in the population size at any given generation, and thus prevent a big loss of genetic diversity in a very short time. Without a shrinkage limit, a small group of “super” designs may appear and quickly cull the population down to a size on the order of the limiting value. In this case, all the diversity of the population is lost and it is expensive to re-diversify and spread the population. The

The replacement_type for a SOGA may be roulette_wheel, unique_roulette_wheel, elitist, or favor_feasible. The elitist selector simply chooses the required number of designs taking the most fit. For example, if 100 selections are requested, then the top 100 designs as ranked by fitness will be selected and the remaining will be discarded. The favor_feasible replacement type first considers feasibility as a selection criteria. If that does not produce a “winner” then it moves on to considering fitness value. Because of this, any fitness assessor used with the favor_feasible selector must only account objectives in the creation of fitness. Therefore, there is such a fitness assessor and it’s use is enforced when the favor_feasible selector is chosen. In that case, and if the output level is set high enough, a message will be presented indicating that the weighted_sum_only fitness assessor will be used.

elitist

- Keywords Area
- method
- soga
- replacement_type
- elitist

Use the best designs to form a new population

Specification

Alias: none

Argument(s): none

Description

The elitist (default) setting creates a new population using (a) the replacement_size best individuals from the current population, (b) and population_size - replacement_size individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default new_solutions_generated value is set such that the entire set of newly generated individuals will be selected for replacement.
favor_feasible

- Keywords Area
- method
- soga
- replacement_type
- favor_feasible

Prioritize feasible designs

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor.

The `favor_feasible` replacement type first considers feasibility as a selection criteria. If that does not produce a ”winner” then it moves on to considering fitness value. Because of this, any fitness assessor used with the `favor_feasible` selector must only account objectives in the creation of fitness. Therefore, there is such a fitness assessor and it’s use is enforced when the `favor_feasible` selector is chosen. In that case, and if the output level is set high enough, a message will be presented indicating that the `weighted_sum_only` fitness assessor will be used.

roulette_wheel

- Keywords Area
- method
- soga
- replacement_type
- roulette_wheel

Replace population

**Specification**

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The replacement_type of roulette_wheel or unique_roulette_wheel may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the layer_rank and domination_count, the recommended selector is the below_limit selector. The below_limit replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The replacement_type of favor_feasible is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).

unique_roulette_wheel

- Keywords Area
- method
- soga
- replacement_type
- unique_roulette_wheel

Replace population

Specification

Alias: none

Argument(s): none

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The replacement_type of roulette_wheel or unique_roulette_wheel may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the layer_rank and domination_count, the recommended selector is the below_limit selector. The below_limit replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The replacement_type of favor_feasible is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).

convergence_type

- Keywords Area
- method
- soga
- convergence_type

Select the convergence type for JEGA methods
### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** average_fitness_tracker

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Group 1</strong></td>
<td></td>
<td>best_fitness_tracker</td>
<td>Tracks the best fitness of the population</td>
</tr>
<tr>
<td></td>
<td></td>
<td>average_fitness_tracker</td>
<td>Tracks the average fitness of the population</td>
</tr>
</tbody>
</table>

### Description

The two JEGA methods use different convergence types, which are described on their respective pages. All the convergence types are modified by the optional keywords `percent_change` and `num_generations`.

**best_fitness_tracker**

- **Keywords Area**
- **method**
- **soga**
- **convergence_type**
- **best_fitness_tracker**

Tracks the best fitness of the population

### Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optional</strong></td>
<td></td>
<td>percent_change</td>
<td>Define the convergence criterion for JEGA methods</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
<td>num_generations</td>
<td>Define the convergence criterion for JEGA methods</td>
</tr>
</tbody>
</table>

### Description

The `best_fitness_tracker` tracks the best fitness in the population. Convergence occurs after `num_generations` has passed and there has been less than `percent_change` in the best fitness value. The percent change can be as low as 0% in which case there must be no change at all over the number of generations.
See Also
These keywords may also be of interest:

- average_fitness_tracker

percent_change

- Keywords Area
- method
- soga
- convergence_type
- best_fitness_tracker
- percent_change

Define the convergence criterion for JEGA methods

Specification

Alias: none
Argument(s): REAL
Default: 0.1

Description

The percent_change is the threshold beneath which convergence is attained whereby it is compared to the metric value computed.

num_generations

- Keywords Area
- method
- soga
- convergence_type
- best_fitness_tracker
- num_generations

Define the convergence criterion for JEGA methods

Specification

Alias: none
Argument(s): INTEGER
Default: 10
CHAPTER 6. KEYWORDS AREA

Description
The `num_generations` is the number of generations over which the metric value should be tracked. Convergence will be attained if the recorded metric is below `percent_change` for `num_generations` consecutive generations.

`average_fitness_tracker`
- **Keywords Area**
- **method**
- **soga**
- **convergence_type**
- **average_fitness_tracker**

Tracks the average fitness of the population

Specification
**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><code>percent_change</code></td>
<td>Define the convergence criterion for JEGA methods</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>num_generations</code></td>
<td>Define the convergence criterion for JEGA methods</td>
</tr>
</tbody>
</table>

Description
The `convergence_type` called `average_fitness_tracker` keeps track of the average fitness in a population. If this average fitness does not change more than `percent_change` over some number of generations, `num_generations`, then the solution is reported as converged and the algorithm terminates.

See Also
These keywords may also be of interest:
- **best_fitness_tracker**

`percent_change`
- **Keywords Area**
- **method**
- **soga**
6.2. METHOD

- convergence_type
- average_fitness_tracker
- percent_change

Define the convergence criterion for JEGA methods

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 0.1

**Description**

The `percent_change` is the threshold beneath which convergence is attained whereby it is compared to the metric value computed.

- `num_generations`
  - **Keywords Area**
  - method
  - soga
  - convergence_type
  - average_fitness_tracker
  - num_generations

Define the convergence criterion for JEGA methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 10

**Description**

The `num_generations` is the number of generations over which the metric value should be tracked. Convergence will be attained if the recorded metric is below `percent_change` for `num_generations` consecutive generations.

**max_iterations**

- **Keywords Area**
- method
- soga
- max_iterations

Stopping criterion based on number of iterations
Topics
This keyword is related to the topics:

- method
- independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

linear_inequality_constraint_matrix

- Keywords Area
- method
- soga
- linear_inequality_constraint_matrix

Define coefficients of the linear inequality constraints

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: no linear inequality constraints

Description
In the inequality case, the constraint matrix $A$ provides coefficients for the variables in the two-sided formulation:

$$a_l ≤ Ax ≤ a_u$$

Where the bounds are optionally specified by linear_inequality_lower_bounds, and linear_-inequality_upper_bounds. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax ≤ 0.0$$
6.2. METHOD

linear_inequality_lower_bounds
  • Keywords Area
  • method
  • soga
  • linear_inequality_lower_bounds

Define lower bounds for the linear inequality constraint

Topics
This keyword is related to the topics:
  • linear_constraints

Specification

Alias: none
Argument(s): REALLIST
Default: vector values = -infinity

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$ a_l \leq Ax \leq a_u $$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$ Ax \leq 0.0 $$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than +bigRealBoundSize (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than -bigRealBoundSize are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since -DBL_MAX < -bigRealBoundSize).

linear_inequality_upper_bounds
  • Keywords Area
  • method
  • soga
  • linear_inequality_upper_bounds

Define upper bounds for the linear inequality constraint
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

• linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 0.

Description
In the inequality case, the lower $a_l$ and upper $a_u$ bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where $A$ is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see objective_functions), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than $+\text{bigRealBoundSize}$ ($1.0e+30$, as defined in Minimizer) are treated as $+\text{infinity}$ and any lower bound values less than $-\text{bigRealBoundSize}$ are treated as $-\text{infinity}$.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since $-\text{DBL\_MAX} < -\text{bigRealBoundSize}$).

linear_inequality_scale_types

• Keywords Area
• method
• soga
• linear_inequality_scale_types

Specify how each linear inequality constraint is scaled

Topics
This keyword is related to the topics:

• linear_constraints

Specification
Alias: none
Argument(s): STRINGLIST
Default: vector values = ”none”
Description

`linear.inequality_scale_types` provide strings specifying the scaling type for each linear inequality constraint, in methods that support scaling. An entry may be selected for each constraint. The options are:

- `'none'` - no scaling
- `'value'` - characteristic value if this is chosen, then `linear.inequality_scales` must be specified
- `'auto'` - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables $x$:

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

`linear.inequality_scales`

- **Keywords Area**
- **method**
- **soga**
- **linear.inequality_scales**

Define the characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- **linear.constraints**

Specification

**Alias:** none

**Argument(s):** REALLIST

**Default:** vector values = 1. (no scaling)
CHAPTER 6. KEYWORDS AREA

Description
Each entry in `linear_inequality_scales` may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

Behavior depends on the choice of `linear_inequality_scale_type`:

- `scale_type` - behavior of `linear_inequality_scales`
- `'none'` - ignored
- `'value'` - required
- `'auto'` - optional

If a single real value is specified it will apply to all components of the constraint.

Scaling for linear constraints is applied after any continuous variable scaling. For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U
\]
\[
a_L \leq A_i (\text{diag}(x_M) \tilde{x} + x_O) \leq a_U
\]
\[
a_L - A_i x_O \leq A_i \text{diag}(x_M) \tilde{x} \leq a_U - A_i x_O
\]
\[
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

`linear_equality_constraint_matrix`

- Keywords Area
- method
- soga
- `linear_equality_constraint_matrix`

Define coefficients of the linear equalities

Topics
This keyword is related to the topics:

- `linear_constraints`

Specification
Alias: none

Argument(s): REALLIST
Default: no linear equality constraints
6.2. METHOD

Description

In the equality case, the constraint matrix $A$ provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

linear_equation_targets

- **Keywords Area**
- **method**
- **soga**
- **linear_equation_targets**

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- **linear_constraints**

Specification

Alias: none

- **Argument(s):** REALLIST
- **Default:** vector values = 0.

Description

In the equality case, the targets $a_t$ provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:

$$Ax = 0.0$$

linear_equation_scale_types

- **Keywords Area**
- **method**
- **soga**
- **linear_equation_scale_types**

Specify how each linear equality constraint is scaled
Topics

This keyword is related to the topics:

- \texttt{linear\_constraints}

Specification

\textbf{Alias:} none

\textbf{Argument(s):} STRINGLIST

\textbf{Default:} vector values = "none"

Description

\texttt{linear\_equality\_scale\_types} provide strings specifying the scaling type for each linear equality constraint, in methods that support scaling.

An entry may be selected for each constraint. The options are:

- \texttt{'none'} - no scaling
- \texttt{'value'} - characteristic value if this is chosen, then \texttt{linear\_equality\_scales} must be specified
- \texttt{'auto'} - automatic scaling If a single string is specified it will apply to all constraints.

Scaling for linear constraints is applied \textit{after} any continuous variable scaling.

For example, for variable scaling on continuous design variables \(x\):

\[
\tilde{x}^j = \frac{x^j - x^j_O}{x^j_M}
\]

we have the following system for linear equality constraints

\[
a_L \leq A_i x \leq a_U \\
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \\
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to \([0,1]\).

\texttt{linear\_equality\_scales}

- Keywords Area
- method
- soga
- \texttt{linear\_equality\_scales}

Define the characteristic values to scale linear equalities
6.2. METHOD

Topics
This keyword is related to the topics:

- linear_constraints

Specification
Alias: none
Argument(s): REALLIST
Default: vector values = 1. (no scaling)

Description
Each entry in linear_equality_scales may be a user-specified, nonzero characteristic value to be used in scaling each constraint.

See the scaling keyword in the method section for details on how to use this keyword.
Scaling for linear constraints is applied after any continuous variable scaling.
For example, for variable scaling on continuous design variables \( x \):

\[
\tilde{x}^j = \frac{x^j - x^O}{x^M}
\]

we have the following system for linear inequality constraints

\[
a_L \leq A_i x \leq a_U \\
a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \\
a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \\
\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U
\]

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

max_function_evaluations
- Keywords Area
- method
- soga
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics
This keyword is related to the topics:

- method_independent_controls
CHAPTER 6. KEYWORDS AREA

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 1000

**Description**

The maximum number of function evaluations (default: 1000). See also `max_iterations`.

**scaling**

- **Keywords Area**
- **method**
- **soga**
- **scaling**

Turn on scaling for variables, responses, and constraints

**Topics**

This keyword is related to the topics:

- **method-independent_controllers**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no scaling

**Description**

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the `method`, `variables`, and `responses` blocks. When the `scaling` keyword is omitted, all `_scale_types` and `*_scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `*scales` keyword, which gives characteristic values
- a `*scale_type` keyword, which determines how to use the characteristic values
The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**
There are four scale types:
1. *none* (default) - no scaling, value of *scales* keyword is ignored
2. *value* - multiplicative scaling
3. *auto* - automatic scaling
   - First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
     - two-sided bounds scaled into the interval [0,1];
     - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
     - no bounds or targets: no automatic scaling possible, therefore no scaling for this component
   - Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. *log* - logarithmic scaling
   - First, any characteristic values from the optional *scales* specification are applied. Then logarithm base 10 scaling is applied.
   - Logarithmic scaling is not available for linear constraints.
   - When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**
The *scales* keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type*, as described above.
Depending on the scale type, the characteristic values may be required or optional.
- *none, auto, log* - optional
- *value* - required.
A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN*. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**
The two examples below are equivalent:
```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
population_size
  • Keywords Area
  • method
  • soga
  • population_size

Set the initial population size in JEGA methods

**Specification**

Alias: none
  Argument(s): INTEGER
  Default: 50

**Description**

The number of designs in the initial population is specified by the `population_size`. Note that the `population_size` only sets the size of the initial population. The population size may vary in the JEGA methods according to the type of operators chosen for a particular optimization run.

log_file
  • Keywords Area
  • method
  • soga
  • log_file

Specify the name of a log file

**Specification**

Alias: none
  Argument(s): STRING
  Default: JEGAGlobal.log

**Description**

New as of JEGA v2.0 is the introduction of the `log_file` specification. JEGA now uses a logging library to output messages and status to the user. JEGA can be configured at build time to log to both the console window and a text file, one or the other, or neither. The `log_file` input is a string name of a file into which to log. If the build was configured without file logging in JEGA, this input is ignored. If file logging is enabled and no `log_file` is specified, the default file name of JEGAGlobal.log is used.
6.2. METHOD

\texttt{print\_each\_pop}

- Keywords Area
- method
- soga
- \texttt{print\_each\_pop}

Print every population to a population file

\textbf{Specification}

\textbf{Alias:} none

\textbf{Argument(s):} none

\textbf{Default:} No printing

\textbf{Description}

New to JEGA v2.0 is the introduction of the \texttt{print\_each\_pop} specification. It serves as a flag and if supplied, the population at each generation will be printed to a file named “population<GEN#>.dat” where <GEN#> is the number of the current generation.

\texttt{initialization\_type}

- Keywords Area
- method
- soga
- \texttt{initialization\_type}

Specify how to initialize the population

\textbf{Specification}

\textbf{Alias:} none

\textbf{Argument(s):} none

\textbf{Default:} unique\_random

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required(Choose One)</td>
<td>Group 1</td>
<td>simple_random</td>
<td>Create random initial solutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique_random</td>
<td>Create random initial solutions, but enforce uniqueness (default)</td>
</tr>
</tbody>
</table>
The initialization_type defines how the initial population is created for the GA. There are three types:

1. simple_random
2. unique_random (default)
3. flat_file

Setting the size for the flat_file initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the unique_random initializer and then the simple_random initializer if necessary.

**simple_random**
- Keywords Area
- method
- soga
- initialization_type
- simple_random

Create random initial solutions

**Specification**

Alias: none
Argument(s): none

**Description**

simple_random creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs.

**unique_random**
- Keywords Area
- method
- soga
- initialization_type
- unique_random

Create random initial solutions, but enforce uniqueness (default)
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

unique_random is the same as simple_random, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected.

flat_file

- Keywords Area
- method
- soga
- initialization_type
- flat_file

Read initial solutions from file

Specification

Alias: none
Argument(s): STRING

Description

flat_file allows the initial population to be read from a flat file. If flat_file is specified, a file name must be given.

Variables can be delimited in the flat file in any way you see fit with a few exceptions. The delimiter must be the same on any given line of input with the exception of leading and trailing whitespace. So a line could look like: 1.1, 2.2, 3.3 for example but could not look like: 1.1, 2.2 3.3. The delimiter can vary from line to line within the file which can be useful if data from multiple sources is pasted into the same input file. The delimiter can be any string that does not contain any of the characters .+-dDeE or any of the digits 0-9. The input will be read until the end of the file. The algorithm will discard any configurations for which it was unable to retrieve at least the number of design variables. The objective and constraint entries are not required but if ALL are present, they will be recorded and the design will be tagged as evaluated so that evaluators may choose not to re-evaluate them.

Setting the size for this initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the unique_random initializer and then the simple_random initializer if necessary.

crossover_type

- Keywords Area
- method
- soga
- crossover_type

Select a crossover type for JEGA methods
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: shuffle_random

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Required (Choose One)</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>Group 1</td>
<td>multi_point_binary</td>
<td>Use bit switching for crossover events</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>multi_point_parameterized_binary</td>
<td>Use bit switching to crossover each design variable</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>multi_point_real</td>
<td>Perform crossover in real valued genome</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>shuffle_random</td>
<td>Perform crossover by choosing design variable(s)</td>
</tr>
</tbody>
</table>

| Required          | crossover_rate | Optional | Specify the probability of a crossover event |

Description

There are many crossover types available. multi_point_binary crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). multi_point_parameterized_binary crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. multi_point_real crossover performs a variable switching crossover routing at N crossover points in the real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

The final crossover type is shuffle_random. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

All crossover types take a crossover_rate. The crossover rate is used to calculate the number of crossover operations that take place. The number of crossovers is equal to the rate * population_size.

multi_point_binary

- Keywords Area
6.2. METHOD

- method
- soga
- crossover_type
- multi_point_binary

Use bit switching for crossover events

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routine at N crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by $10^6$ and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

`multi_point_parameterized_binary`

- **Keywords** Area
- method
- soga
- crossover_type
- multi_point_parameterized_binary

Use bit switching to crossover each design variable

**Specification**

**Alias:** none

**Argument(s):** INTEGER
CHAPTER 6. KEYWORDS AREA

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by $10^6$ and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

`multi_point_real`

- Keywords Area
- method
- soga
- crossover_type
- `multi_point_real`

Perform crossover in real valued genome

Specification

Alias: none

Argument(s): INTEGER

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by $10^6$ and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.
6.2. METHOD

shuffle_random

- Keywords Area
- method
- soga
- crossover_type
- shuffle_random

Perform crossover by choosing design variable(s)

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_parents</td>
<td>num_offspring</td>
<td>Number of parents in random shuffle crossover</td>
</tr>
<tr>
<td>Optional</td>
<td>num_offspring</td>
<td></td>
<td>Number of offspring in random shuffle crossover</td>
</tr>
</tbody>
</table>

Description

The final crossover type is shuffle_random. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

num_parents

- Keywords Area
- method
- soga
- crossover_type
- shuffle_random
- num_parents

Number of parents in random shuffle crossover
**Specification**

Alias: none

- Argument(s): INTEGER
- Default: 2

**Description**

Number of parents in random shuffle crossover

\[
\text{num_offspring}
\]

- Keywords Area
- method
- soga
- crossover_type
- shuffle_random
- num_offspring

Number of offspring in random shuffle crossover

**Specification**

Alias: none

- Argument(s): INTEGER
- Default: 2

**Description**

Number of offspring in random shuffle crossover

\[
\text{crossover_rate}
\]

- Keywords Area
- method
- soga
- crossover_type
- crossover_rate

Specify the probability of a crossover event

**Specification**

Alias: none

- Argument(s): REAL
- Default: 0.8
6.2. METHOD

Description

The `crossover_type` controls what approach is employed for combining parent genetic information to create offspring, and the `crossover_rate` specifies the probability of a crossover operation being performed to generate a new offspring. The SCOLIB EA method supports three forms of crossover, `two_point`, `blend`, and `uniform`, which generate a new individual through combinations of two parent individuals. Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate. Uniform crossover creates offspring through random combination of coordinates from the two parents. Blend crossover generates a new individual randomly along the multidimensional vector connecting the two parents.

`mutation_type`

- Keywords Area
- `method`
- `soga`
- `mutation_type`

Select a mutation type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: `replace_uniform`

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> (Choose One)</td>
<td>Group 1</td>
<td><code>replace_uniform</code></td>
<td>Mutate by flipping a random bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>offset_normal</code></td>
<td>Set mutation offset to use a normal distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>offset_cauchy</code></td>
<td>Use a Cauchy distribution for the mutation offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>offset_uniform</code></td>
<td>Set mutation offset to use a uniform distribution</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

**Description**

Five mutation types are available for selection by keyword: `replace_uniform`, `bit_random`, `offset_cauchy`, `offset_normal`, and `offset_uniform`. They are described in greater detail on their respective keyword pages.

The `offset_*` mutators all act by adding a random "offset" to a variable value. The random amount has a mean of zero in all cases. The size of the offset is controlled using the `mutation_scale` keyword, which is interpreted differently for each `offset_*` type.

The rate of mutations for all types is controlled using the `mutation_rate`. The rate is applied differently in each `mutation_type`.

**bit_random**

- Keywords Area
- method
- soga
- mutation_type
- bit_random

Mutate by flipping a random bit

**Specification**

Alias: none

Argument(s): none

**Description**

The `bit_random` mutator introduces random variation by first converting a randomly chosen variable of a randomly chosen design into a binary string. It then flips a randomly chosen bit in the string from a 1 to a 0 or visa versa. In this mutation scheme, the resulting value has more probability of being similar to the original value.

**replace_uniform**

- Keywords Area
- method
- soga
- mutation_type
- replace_uniform

Use uniformly distributed value over range of parameter
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

replace_uniform introduces random variation by first randomly choosing a design variable of a randomly selected design and reassigning it to a random valid value for that variable. No consideration of the current value is given when determining the new value.

offset_normal

- Keywords Area
- method
- soga
- mutation_type
- offset_normal

Set mutation offset to use a normal distribution

 Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>mutation_scale</td>
<td>Description</td>
</tr>
</tbody>
</table>

| Description |

The offset_normal mutator introduces random variation by adding a Gaussian random amount to a variable value. The random amount has a standard deviation dependent on the mutation_scale.

mutation_scale

- Keywords Area
- method
- soga
- mutation_type
- offset_normal
- mutation_scale

Scales mutation across range of parameter
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): REAL
Default: 0.15

Description

The mutation_scale is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected mutation_type. For offset_normal and offset_cauchy, mutation_scale is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For offset_uniform, the range of possible deviation amounts is +/- 1/2 * (mutation_scale * variable range).

offset_cauchy

- Keywords Area
- method
- soga
- mutation_type
- offset_cauchy

Use a Cauchy distribution for the mutation offset

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>mutation_scale</td>
<td>Scales mutation across range of parameter</td>
</tr>
</tbody>
</table>

Description

The offset_cauchy mutator introduces random variation by adding a Cauchy random amount to a variable value. The random amount has a standard deviation dependent on the mutation_scale.

mutation_scale

- Keywords Area
- method
- soga
- mutation_type
- offset_cauchy
6.2. METHOD

- **mutation_scale**

  Scales mutation across range of parameter

**Specification**

*Alias:* none

*Argument(s):* REAL

*Default:* 0.15

**Description**

The mutation_scale is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected mutation_type. For offset_normal and offset_cauchy, mutation_scale is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For offset_uniform, the range of possible deviation amounts is +/- 1/2 * (mutation_scale * variable range).

**offset_uniform**

- **Keywords Area**

- **method**

- **soga**

- **mutation_type**

- **offset_uniform**

  Set mutation offset to use a uniform distribution

**Specification**

*Alias:* none

*Argument(s):* none

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>mutation_scale</td>
<td>Scales mutation across range of parameter</td>
</tr>
</tbody>
</table>

**Description**

The offset_uniform mutator introduces random variation by adding a uniform random amount to a variable value. The random amount depends on the mutation_scale.
mutation_scale
• Keywords Area
• method
• soga
• mutation_type
• offset_uniform
• mutation_scale
Scales mutation across range of parameter

**Specification**

Alias: none
Argument(s): REAL
Default: 0.15

**Description**

The mutation_scale is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected mutation_type. For offset_normal and offset_cauchy, mutation_scale is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For offset_uniform, the range of possible deviation amounts is +/- 1/2 * (mutation_scale * variable range).

mutation_rate
• Keywords Area
• method
• soga
• mutation_type
• mutation_rate
Set probability of a mutation

**Specification**

Alias: none
Argument(s): REAL
Default: 0.08

**Description**

All mutation types have a mutation_rate, which controls the number of mutations performed. For replace_uniform and all the offset_* types, the number of mutations performed is the product of mutation_rate and population_size. For bit_random, it’s the product of the mutation_rate, number of design variables, and population size.
6.2. METHOD

seed

- Keywords Area
- method
- soga
- seed

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)

**Description**

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**convergence_tolerance**

- Keywords Area
- method
- soga
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: 1.e-4

Description
The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library
This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:
- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

model_pointer

- Keywords Area
- method
- soga
- model_pointer

Identifier for model block to be used by a method
6.2. METHOD

Topics

This keyword is related to the topics:

- block_pointer

Specification

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```
environment
   tabular_graphics_data
   method_pointer = 'UQ'

method
   id_method = 'UQ'
   model_pointer = 'SURR'
   sampling,
      samples = 10
      seed = 98765 rng rnum2
      response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
      0.1 0.2 0.6
      sample_type lhs
      distribution cumulative

model
   id_model = 'SURR'
   surrogate global,
   dace_method_pointer = 'DACE'
   polynomial quadratic

method
   id_method = 'DACE'
   model_pointer = 'DACE_M'
   sampling sample_type lhs
      samples = 121 seed = 5034 rng rnum2

model
   id_model = 'DACE_M'
   single
```
interface_pointer = 'I1'

variables
    uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
    system async evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
    response_functions = 3
    no_gradients
    no_hessians

6.2.34 coliny_pattern_search

- Keywords Area
- method
- coliny_pattern_search

Pattern search, derivative free optimization method

Topics

This keyword is related to the topics:

- package_scolib
- package_coliny
- global_optimization_methods

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>constant_penalty</td>
<td>constant_penalty</td>
<td>Use a simple weighted penalty to manage feasibility</td>
</tr>
</tbody>
</table>
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no_expansion</td>
<td>Don’t allow expansion of the search pattern</td>
</tr>
<tr>
<td></td>
<td>expand_after_success</td>
<td>Set the factor by which a search pattern can be expanded</td>
</tr>
<tr>
<td></td>
<td>pattern_basis</td>
<td>Pattern basis selection</td>
</tr>
<tr>
<td></td>
<td>stochastic</td>
<td>Generate trial points in random order</td>
</tr>
<tr>
<td></td>
<td>total_pattern_size</td>
<td>Total number of points in search pattern</td>
</tr>
<tr>
<td></td>
<td>exploratory_moves</td>
<td>Exploratory moves selection</td>
</tr>
<tr>
<td></td>
<td>synchronization</td>
<td>Select how Dakota schedules function evaluations in a pattern search</td>
</tr>
<tr>
<td></td>
<td>contraction_factor</td>
<td>Amount by which step length is rescaled</td>
</tr>
<tr>
<td></td>
<td>constraint_penalty</td>
<td>Multiplier for the penalty function</td>
</tr>
<tr>
<td></td>
<td>initial_delta</td>
<td>Initial step size for non-gradient based optimizers</td>
</tr>
<tr>
<td></td>
<td>threshold_delta</td>
<td>Stopping criteria based on step length or pattern size</td>
</tr>
<tr>
<td></td>
<td>solution_target</td>
<td>Stopping criteria based on objective function value</td>
</tr>
<tr>
<td></td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td></td>
<td>show_misc_options</td>
<td>Show algorithm parameters not exposed in Dakota input</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| Optional | misc_options | Set method options not available through Dakota spec |
| Optional | max_iterations | Stopping criterion based on number of iterations |
| Optional | convergence_tolerance | Stopping criterion based on convergence of the objective function or statistics |
| Optional | max_function_evaluations | Stopping criteria based on number of function evaluations |
| Optional | scaling | Turn on scaling for variables, responses, and constraints |
| Optional | model_pointer | Identifier for model block to be used by a method |

**Description**

Pattern search techniques are nongradient-based optimization methods which use a set of offsets from the current iterate to locate improved points in the design space.

See the page package_scolib for important information regarding all SCOLIB methods

coliny_pattern_search supports concurrency up to the size of the search pattern

Traditional pattern search methods search with a fixed pattern of search directions to try to find improvements to the current iterate. The SCOLIB pattern search methods generalize this simple algorithmic strategy to enable control of how the search pattern is adapted, as well as how each search pattern is evaluated. The stochastic and synchronization specifications denote how the the trial points are evaluated. The stochastic specification indicates that the trial points are considered in a random order. For parallel pattern search, synchronization dictates whether the evaluations are scheduled using a blocking scheduler or a nonblocking scheduler. In the blocking case, all points in the pattern are evaluated (in parallel), and if the best of these trial points is an improving point, then it becomes the next iterate. These runs are reproducible, assuming use of the same seed in the stochastic case. In the nonblocking case, all points in the pattern may not be evaluated, since the first improving point found becomes the next iterate. Since the algorithm steps will be subject to parallel timing variabilities, these runs will not generally be repeatable. The synchronization specification has similar connotations for sequential pattern search. If blocking is specified, then each sequential iteration terminates after all trial points have been considered, and if nonblocking is specified, then each sequential iteration terminates after the first improving trial point is evaluated. In this release, both blocking and nonblocking specifications result in blocking behavior (except in the case where exploratory_moves below is set to adaptive_pattern). Nonblocking behavior will be re-enabled after some underlying technical issues have been resolved.
The particular form of the search pattern is controlled by the `pattern_basis` specification. If `pattern_basis` is coordinate basis, then the pattern search uses a plus and minus offset in each coordinate direction, for a total of $2n$ function evaluations in the pattern. This case is depicted in Figure 5.3 for three coordinate dimensions.

![Figure 6.1: Depiction of coordinate pattern search algorithm](image)

If `pattern_basis` is simplex, then pattern search uses a minimal positive basis simplex for the parameter space, for a total of $n+1$ function evaluations in the pattern. Note that the simplex pattern basis can be used for unbounded problems only. The `total_pattern_size` specification can be used to augment the basic coordinate and simplex patterns with additional function evaluations, and is particularly useful for parallel load balancing. For example, if some function evaluations in the pattern are dropped due to duplication or bound constraint interaction, then the `total_pattern_size` specification instructs the algorithm to generate new offsets to bring the total number of evaluations up to this consistent total.

The `exploratory_moves` specification controls how the search pattern is adapted. (The search pattern can be adapted after an improving trial point is found, or after all trial points in a search pattern have been found to be unimproving points.) The following exploratory moves selections are supported by SCOLIB:

- The `basic_pattern` case is the simple pattern search approach, which uses the same pattern in each iteration.

- The `multi_step` case examines each trial step in the pattern in turn. If a successful step is found, the pattern search continues examining trial steps about this new point. In this manner, the effects of multiple
successful steps are cumulative within a single iteration. This option does not support any parallelism and will result in a serial pattern search.

• The **adaptive_pattern** case invokes a pattern search technique that adaptively rescales the different search directions to maximize the number of redundant function evaluations. See\[45\] for details of this method. In preliminary experiments, this method had more robust performance than the standard **basic_pattern** case in serial tests. This option supports a limited degree of parallelism. After successful iterations (where the step length is not contracted), a parallel search will be performed. After unsuccessful iterations (where the step length is contracted), only a single evaluation is performed.

The **initial_delta** and **threshold_delta** specifications provide the initial offset size and the threshold size at which to terminate the algorithm. For any dimension that has both upper and lower bounds, this step length will be internally rescaled to provide search steps of length \(\text{initial}_\text{delta} \times \text{range} \times 0.1\). This rescaling does not occur for other dimensions, so search steps in those directions have length **initial_delta**. Note that the factor of 0.1 in the rescaling could result in an undesirably small initial step. This can be offset by providing a large **initial_delta**.

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value **contraction_factor**, and they expand the step length by the value \((1/\text{contraction}\_\text{factor})\). The **expand_after_success** control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the **no_expansion** flag instructs the algorithm to forgo pattern expansion altogether.

Finally, constraint infeasibility can be managed in a somewhat more sophisticated manner than the simple weighted penalty function. If the **constant_penalty** specification is used, then the simple weighted penalty scheme described above is used. Otherwise, the constraint penalty is adapted to the value \(\text{constraint}\_\text{penalty}/L\), where \(L\) is the the smallest step length used so far.

**See Also**

These keywords may also be of interest:

• **coliny_beta**
• **coliny_direct**
• **coliny_cobyla**
• **coliny_ea**
• **coliny_solis_wets**

**constant_penalty**

• **Keywords Area**
• **method**
• **coliny_pattern_search**
• **constant_penalty**

Use a simple weighted penalty to manage feasibility
6.2. METHOD

Specification

Alias: none

Argument(s): none

Default: algorithm dynamically adapts the constraint penalty

Description

Finally, constraint infeasibility can be managed in a somewhat more sophisticated manner than the simple weighted penalty function. If the constant_penalty specification is used, then the simple weighted penalty scheme described above is used. Otherwise, the constraint penalty is adapted to the value constraint_penalty/L, where L is the the smallest step length used so far.

no_expansion

- Keywords Area
- method
- coliny_pattern_search
- no_expansion

Don’t allow expansion of the search pattern

Specification

Alias: none

Argument(s): none

Default: algorithm may expand pattern size

Description

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value contraction_factor, and they expand the step length by the value (1/contraction_factor). The expand_after_success control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the no_-expansion flag instructs the algorithm to forgo pattern expansion altogether.

expand_after_success

- Keywords Area
- method
- coliny_pattern_search
- expand_after_success

Set the factor by which a search pattern can be expanded
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGER

Default: 5

Description

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value contraction_factor, and they expand the step length by the value (1/contraction_factor). The expand_after_success control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the no_-expansion flag instructs the algorithm to forgo pattern expansion altogether.

pattern_basis

- Keywords Area
- method
- coliny_pattern_search
- pattern_basis

Pattern basis selection

Specification

Alias: none

Argument(s): none

Default: coordinate

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword coordinate</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>coordinate</td>
<td>Use coordinate directions as search pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td>simplex</td>
<td>Use a minimal simplex for the search pattern</td>
</tr>
</tbody>
</table>

Description

The particular form of the search pattern is controlled by the pattern_basis specification. If pattern_-basis is coordinate basis, then the pattern search uses a plus and minus offset in each coordinate direction, for a total of 2n function evaluations in the pattern. This case is depicted in Figure 5.3 for three coordinate dimensions.

coordinate

- Keywords Area
- method
6.2. METHOD

- coliny_pattern_search
- pattern_basis
- coordinate

Use coordinate directions as search pattern

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The particular form of the search pattern is controlled by the pattern_basis specification. If pattern_basis is coordinate basis, then the pattern search uses a plus and minus offset in each coordinate direction, for a total of \(2n\) function evaluations in the pattern. This case is depicted in Figure 5.3 for three coordinate dimensions.

**simplex**

- Keywords Area
- method
- coliny_pattern_search
- pattern_basis
- simplex

Use a minimal simplex for the search pattern

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

If pattern_basis is simplex, then pattern search uses a minimal positive basis simplex for the parameter space, for a total of \(n+1\) function evaluations in the pattern. Note that the simplex pattern basis can be used for unbounded problems only. The total_pattern_size specification can be used to augment the basic coordinate and simplex patterns with additional function evaluations, and is particularly useful for parallel load balancing. For example, if some function evaluations in the pattern are dropped due to duplication or bound constraint interaction, then the total_pattern_size specification instructs the algorithm to generate new offsets to bring the total number of evaluations up to this consistent total.
stochastic
- Keywords Area
- method
- coliny_pattern_search
- stochastic

Generate trial points in random order

**Specification**

Alias: none

Argument(s): none

**Description**

Traditional pattern search methods search with a fixed pattern of search directions to try to find improvements to the current iterate. The SCOLIB pattern search methods generalize this simple algorithmic strategy to enable control of how the search pattern is adapted, as well as how each search pattern is evaluated. The stochastic and synchronization specifications denote how the trial points are evaluated. The stochastic specification indicates that the trial points are considered in a random order. For parallel pattern search, synchronization dictates whether the evaluations are scheduled using a blocking scheduler or a nonblocking scheduler (i.e.,

**total_pattern_size**

- Keywords Area
- method
- coliny_pattern_search
- total_pattern_size

Total number of points in search pattern

**Specification**

Alias: none

Argument(s): INTEGER

Default: no augmentation of basic pattern

**Description**

If pattern_basis is simplex, then pattern search uses a minimal positive basis simplex for the parameter space, for a total of $n+1$ function evaluations in the pattern. Note that the simplex pattern basis can be used for unbounded problems only. The total_pattern_size specification can be used to augment the basic coordinate and simplex patterns with additional function evaluations, and is particularly useful for parallel load balancing. For example, if some function evaluations in the pattern are dropped due to duplication or bound constraint interaction, then the total_pattern_size specification instructs the algorithm to generate new offsets to bring the total number of evaluations up to this consistent total.
6.2. \textit{METHOD}

exploratory\_moves
- Keywords Area
- method
- coliny\_pattern\_search
- exploratory\_moves

Exploratory moves selection

\textbf{Specification}

\textbf{Alias:} none  
\textbf{Argument(s):} none  
\textbf{Default:} basic\_pattern

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required\textit{(Choose One)}</td>
<td>Group 1</td>
<td>multi_step</td>
<td>Examine trial step around successful new point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adaptive_pattern</td>
<td>Adaptively rescale search directions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic_pattern</td>
<td>Use the same search pattern every iteration</td>
</tr>
</tbody>
</table>

\textbf{Description}

The \texttt{exploratory\_moves} specification controls how the search pattern is adapted. (The search pattern can be adapted after an improving trial point is found, or after all trial points in a search pattern have been found to be unimproving points.) The following exploratory moves selections are supported by SCOLIB:

\texttt{multi\_step}
- Keywords Area
- method
- coliny\_pattern\_search
- exploratory\_moves
- multi\_step

Examine trial step around successful new point

\textbf{Specification}

\textbf{Alias:} none  
\textbf{Argument(s):} none
CHAPTER 6. KEYWORDS AREA

Description

The **multi_step** case examines each trial step in the pattern in turn. If a successful step is found, the pattern search continues examining trial steps about this new point. In this manner, the effects of multiple successful steps are cumulative within a single iteration. This option does not support any parallelism and will result in a serial pattern.

**adaptive_pattern**

- Keywords Area
- method
- coliny_pattern_search
- exploratory_moves
- adaptive_pattern

Adaptively rescale search directions

Specification

Alias: none
- Argument(s): none

Description

The **adaptive_pattern** case invokes a pattern search technique that adaptively rescales the different search directions to maximize the number of redundant function evaluations. See[45] for details of this method. In preliminary experiments, this method had more robust performance than the standard **basic_pattern** case in serial tests. This option supports a limited degree of parallelism. After successful iterations (where the step length is not contracted), a parallel search will be performed. After unsuccessful iterations (where the step length is contracted), only a single evaluation is performed.

**basic_pattern**

- Keywords Area
- method
- coliny_pattern_search
- exploratory_moves
- basic_pattern

Use the same search pattern every iteration

Specification

Alias: none
- Argument(s): none
6.2. METHOD

Description

The basic_pattern case is the simple pattern search approach, which uses the same pattern in each iteration.

synchronization

- Keywords Area
- method
- coliny_pattern_search
- synchronization

Select how Dakota schedules function evaluations in a pattern search

Specification

Alias: none
Argument(s): none
Default: nonblocking

<table>
<thead>
<tr>
<th>Required-/Optional</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocking</td>
<td></td>
<td>nonblocking</td>
<td>Evaluate points in the pattern until an improving point is found</td>
</tr>
</tbody>
</table>

Description

The synchronization specification can be used to specify the use of either blocking or nonblocking schedulers.

blocking

- Keywords Area
- method
- coliny_pattern_search
- synchronization
- blocking

Evaluate all points in a pattern

Specification

Alias: none
Argument(s): none
**Description**

In the *blocking* case, all points in the pattern are evaluated (in parallel), and if the best of these trial points is an improving point, then it becomes the next iterate. These runs are reproducible, assuming use of the same seed in the stochastic case.

**nonblocking**

- **Keywords Area**
- **method**
- **coliny_pattern_search**
- **synchronization**
- **nonblocking**

Evaluate points in the pattern until an improving point is found

**Specification**

Alias: none

**Argument(s):** none

**Description**

In the *nonblocking* case, all points in the pattern may not be evaluated. The first improving point found becomes the next iterate. Since the algorithm steps will be subject to parallel timing variabilities, these runs will not generally be repeatable.

**contraction_factor**

- **Keywords Area**
- **method**
- **coliny_pattern_search**
- **contraction_factor**

Amount by which step length is rescaled

**Specification**

Alias: none

**Argument(s):** REAL

**Default:** 0.5

**Description**

For pattern search methods, *contraction_factor* specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction\_factor}$
6.2. METHOD

**constraint_penalty**

- **Keywords Area**
- **method**
- **coliny_pattern_search**
- **constraint_penalty**

Multiplier for the penalty function

**Specification**

**Alias**: none
- **Argument(s)**: REAL
- **Default**: 1.0

**Description**

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

**initial_delta**

- **Keywords Area**
- **method**
- **coliny_pattern_search**
- **initial_delta**

Initial step size for non-gradient based optimizers

**Specification**

**Alias**: none
- **Argument(s)**: REAL
- **Default**: 1.0 (COBYLA), 0.1*range (PS, SW)

**Description**

If `initial_delta` is supplied by the user, it will be applied in an absolute sense in all coordinate directions. APPS documentation advocates choosing `initial_delta` to be the approximate distance from the initial point to the solution. If this is unknown, it is advisable to err on the side of choosing an `initial_delta` that is too large or to not specify it. In the latter case, APPS will take a full step to the boundary in each direction. Relative application of `initial_delta` is not available unless the user scales the problem accordingly.
threshold _delta
   
   - Keywords Area
   - method
   - coliny_pattern_search
   - threshold_delta

Stopping criteria based on step length or pattern size

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** 1.0e-4 (COBYLA), 1.0e-5 (PS), 1.0e-6 (SW)

**Description**

threshold _delta is the step length or pattern size used to determine convergence.

**solution_target**

- Keywords Area
- method
- coliny_pattern_search
- solution_target

Stopping criteria based on objective function value

**Specification**

**Alias:** solution_accuracy  
**Argument(s):** REAL  
**Default:** -DBL_MAX

**Description**

solution_target is a termination criterion. The algorithm will terminate when the function value falls below solution_target.

**seed**

- Keywords Area
- method
- coliny_pattern_search
- seed

Seed of the random number generator
6.2. METHOD

**Specification**

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

**Description**

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

show_misc_options
```

- Keywords Area
- method
- coliny_pattern_search
- show_misc_options

Show algorithm parameters not exposed in Dakota input

**Specification**

Alias: none

Argument(s): none

Default: no dump of specification options

**Description**

All SCOLIB methods support the show_misc_options optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The misc_options optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using
this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

**misc_options**

- **Keywords Area**
- **method**
- **coliny_pattern_search**
- **misc_options**

Set method options not available through Dakota spec

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** no misc options

**Description**

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

**max_iterations**

- **Keywords Area**
- **method**
- **coliny_pattern_search**
- **max_iterations**

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**
6.2. METHOD

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

corvergence_tolerance

- Keywords Area
- method
- coliny_pattern_search
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

NL2SOL: See nl2sol

max_function_evaluations

Keywords Area
method
coliny_pattern_search
max_function_evaluations

Stopping criteria based on number of function evaluations

Topics
This keyword is related to the topics:

method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max_iterations.

calining

Keywords Area
method
coliny_pattern_search
scaling

Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:

method_independent_controls
6.2. METHOD

Specification

Alias: none
    Argument(s): none
    Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.
Scales

The *scales* keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type*, as described above. Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0 \times 10^{10}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

model_pointer

- Keywords Area
- method
- coliny_pattern_search
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
6.2. **METHOD**

**Description**

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
    tabular_graphics_data
    method_pointer = 'UQ'

method
    id_method = 'UQ'
    model_pointer = 'SURR'
    sampling,
        samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
    sample_type lhs
distribution cumulative

model
    id_model = 'SURR'
    surrogate global,
        dace_method_pointer = 'DACE'
        polynomial quadratic

method
    id_method = 'DACE'
    model POINTER = 'DACE_M'
    sampling sample_type lhs
        samples = 121 seed = 5034 rng rnum2

model
    id_model = 'DACE_M'
    single
    interface_pointer = 'I1'

variables
    uniform_uncertain = 2
        lower_bounds = 0. 0.
        upper_bounds = 1. 1.
        descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
```
6.2.35  coliny_solis_wets

- **Keywords Area**
- method
- coliny_solis_wets

Simple greedy local search method

Topics

This keyword is related to the topics:

- package_scolib
- package_coliny

Specification

Alias: none

<table>
<thead>
<tr>
<th>Argument(s): none</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>contract_after_/failure</td>
<td></td>
<td>The number of unsuccessful cycles prior to contraction. Don’t allow expansion of the search pattern Set the factor by which a search pattern can be expanded Use a simple weighted penalty to manage feasibility Amount by which step length is rescaled</td>
</tr>
<tr>
<td>Optional</td>
<td>noExpansion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>expand_after_/success</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>constant_penalty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>contraction_factor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_penalty</td>
<td>Multiplier for the penalty function</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>initial_delta</td>
<td>Initial step size for non-gradient based optimizers</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>threshold_delta</td>
<td>Stopping criteria based on step length or pattern size</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>solution_target</td>
<td>Stopping criteria based on objective function value</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>Seed of the random number generator</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>show_misc_options</td>
<td>Show algorithm parameters not exposed in Dakota input</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>misc_options</td>
<td>Set method options not available through Dakota spec</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| Optional | model_pointer | Identifier for model block to be used by a method |

**Description**

The Solis-Wets method is a simple greedy local search heuristic for continuous parameter spaces. Solis-Wets generates trial points using a multivariate normal distribution, and unsuccessful trial points are reflected about the current point to find a descent direction.

See the page package_scolib for important information regarding all SCOLIB methods coliny.solis.wets is inherently serial, no concurrency is used.

These specifications have the same meaning as corresponding specifications for coliny_pattern_search. Please see that page for specification details.

In particular, coliny_solis_wets supports dynamic rescaling of the step length, and dynamic rescaling of the constraint penalty. The only new specification is contract_after_failure, which specifies the number of unsuccessful cycles which must occur with a specific delta prior to contraction of the delta.

**See Also**

These keywords may also be of interest:

- coliny_beta
- coliny_direct
- coliny_pattern_search
- coliny_cobyla
- coliny_ea

**contract_after_failure**

- Keywords Area
- method
- coliny_solis_wets
- contract_after_failure

The number of unsuccessful cycles prior to contraction.

**Specification**

Alias: none

Argument(s): INTEGER

Default: 4*number of variables

**Description**

In particular, coliny_solis_wets supports dynamic rescaling of the step length, and dynamic rescaling of the constraint penalty. The only new specification is contract_after_failure, which specifies the number of unsuccessful cycles which must occur with a specific delta prior to contraction of the delta.
6.2. METHOD

**no_expansion**

- **Keywords Area**
- **method**
- **coliny_solis_wets**
- **no_expansion**

Don’t allow expansion of the search pattern

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** algorithm may expand pattern size

**Description**

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value `contraction_factor`, and they expand the step length by the value `1/contraction_factor`. The `expand_after_success` control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the `no_expansion` flag instructs the algorithm to forgo pattern expansion altogether.

**expand_after_success**

- **Keywords Area**
- **method**
- **coliny_solis_wets**
- **expand_after_success**

Set the factor by which a search pattern can be expanded

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 5

**Description**

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value `contraction_factor`, and they expand the step length by the value `1/contraction_factor`. The `expand_after_success` control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the `no_expansion` flag instructs the algorithm to forgo pattern expansion altogether.
constant_penalty

- Keywords Area
- method
- coliny_solis_wets
- constant_penalty

Use a simple weighted penalty to manage feasibility

**Specification**

**Alias**: none

**Argument(s)**: none

**Default**: algorithm dynamically adapts the constraint penalty

**Description**

Finally, constraint infeasibility can be managed in a somewhat more sophisticated manner than the simple weighted penalty function. If the `constant_penalty` specification is used, then the simple weighted penalty scheme described above is used. Otherwise, the constraint penalty is adapted to the value `constraint_penalty/L`, where $L$ is the the smallest step length used so far.

**contraction_factor**

- Keywords Area
- method
- coliny_solis_wets
- contraction_factor

Amount by which step length is rescaled

**Specification**

**Alias**: none

**Argument(s)**: REAL

**Default**: 0.5

**Description**

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1. For methods that can expand the step length, the expansion is $1/contraction_factor$.
6.2. METHOD

constraint_penalty

- Keywords Area
- method
- coliny_solis_wets
- constraint_penalty

Multiplier for the penalty function

Specification

Alias: none
Argument(s): REAL
Default: 1.0

Description

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds constraint_penalty times the sum of squares of the constraint violations to the objective function. The default value of constraint_penalty is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

initial_delta

- Keywords Area
- method
- coliny_solis_wets
- initial_delta

Initial step size for non-gradient based optimizers

Specification

Alias: none
Argument(s): REAL
Default: 1.0 (COBYLA), 0.1*range (PS, SW)

Description

If initial_delta is supplied by the user, it will be applied in an absolute sense in all coordinate directions. APPS documentation advocates choosing initial_delta to be the approximate distance from the initial point to the solution. If this is unknown, it is advisable to err on the side of choosing an initial_delta that is too large or to not specify it. In the latter case, APPS will take a full step to the boundary in each direction. Relative application of initial_delta is not available unless the user scales the problem accordingly.
threshold_delta

- Keywords Area
- method
- coliny_solis_wets
- threshold_delta

Stopping criteria based on step length or pattern size

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** 1.0e-4 (COBYLA), 1.0e-5 (PS), 1.0e-6 (SW)

**Description**

`threshold_delta` is the step length or pattern size used to determine convergence.

solution_target

- Keywords Area
- method
- coliny_solis_wets
- solution_target

Stopping criteria based on objective function value

**Specification**

**Alias:** solution_accuracy  
**Argument(s):** REAL  
**Default:** -DBL_MAX

**Description**

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

seed

- Keywords Area
- method
- coliny_solis_wets
- seed

Seed of the random number generator
6.2. METHOD

Specification

Alias: none
Argument(s): INTEGER
Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

show_misc_options

  • Keywords Area
  • method
  • coliny_solis_wets
  • show_misc_options

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none
Argument(s): none
Default: no dump of specification options

Description

All SCOLIB methods support the show_misc_options optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The misc_options optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using
this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

**misc_options**

- Keywords Area
- method
- coliny_solis_wets
- misc_options

Set method options not available through Dakota spec

**Specification**

**Alias:** none  
**Argument(s):** STRINGLIST  
**Default:** no misc options

**Description**

All SCOLIB methods support the show_misc_options optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The misc_options optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

**max_iterations**

- Keywords Area
- method
- coliny_solis_wets
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls
6.2. METHOD

Specification

Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global {reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- coliny_solis_wets
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
Argument(s): REAL
Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
• NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

• NL2SOL: See nl2sol

max_function_evaluations

• Keywords Area

• method

• coliny_solis_wets

• max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

• method_independent_controls

Specification

Alias: none
Argument(s): INTEGER
Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling

• Keywords Area

• method

• coliny_solis_wets

• scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

• method_independent_controls
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the `method`, `variables`, and `responses` blocks. When the `scaling` keyword is omitted, all `*scales` and `*_scales` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- `a`∗`scales` keyword, which gives characteristic values
- `a`∗`scale_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. **none** (default) - no scaling, value of `*scales` keyword is ignored
2. **value** - multiplicative scaling
3. **auto** - automatic scaling
4. **log** - logarithmic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. **log** - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.
Scales

The *scales* keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type*, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0\times10^{10}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```plaintext
responses
    objective_functions 3
    sense "maximize"
    primary_scale_types = "value"
    primary_scales = 1 1 100

responses
    objective_functions 3
    sense "maximize"
    primary_scale_types = "value" "value" "value"
    primary_scales = 1 1 100
```

model_pointer

- **Keywords Area**
- **method**
- **coliny_solis_wets**
- **model_pointer**

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- **block_pointer**

Specification

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The **model_pointer** is used to specify which **model** block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a **model** block in the Dakota input file that has a corresponding **id_model** with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a **model_pointer** for each method is imperative.

See **block_pointer** for details about pointers.

**Examples**

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SRR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SRR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
```
response_functions = 3
no_gradients
no_hessians

6.2.36 coliny_cobyla

- Keywords Area
- method
- coliny_cobyla

Constrained Optimization BY Linear Approximations (COBYLA)

Topics

This keyword is related to the topics:

- package_scolib
- package_coliny
- local_optimization_methods
- constrained

Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>initial_delta</td>
<td>Reasonable initial changes to optimization variables</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>threshold_delta</td>
<td>Required or expected accuracy in optimization variables.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>solution_target</td>
<td>Stopping criteria based on objective function value</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>show_misc_options</th>
<th>Show algorithm parameters not exposed in Dakota input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>misc_options</td>
<td>Set method options not available through Dakota spec</td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

The Constrained Optimization BY Linear Approximations (COBYLA) algorithm is an extension to the Nelder-Mead simplex algorithm for handling general linear/nonlinear constraints and is invoked using the `coliny_cobyla` group specification. The COBYLA algorithm employs linear approximations to the objective and constraint functions, the approximations being formed by linear interpolation at N+1 points in the space of the variables. We regard these interpolation points as vertices of a simplex. The step length parameter controls the size of the simplex and it is reduced automatically from `initial_delta` to `threshold_delta`. One advantage that COBYLA has over many of its competitors is that it treats each constraint individually when calculating a change to the variables, instead of lumping the constraints together into a single penalty function.

*See the page package_scolib for important information regarding all SCOLIB methods*

`coliny_cobyla` is inherently serial.

**Stopping Criteria**

COBYLA currently only supports termination based on

- `max_function_evaluations`
- `solution_target`

Other method-independent stopping criteria (`max_iterations` and `convergence_tolerance`) will be ignored if set.
See Also

These keywords may also be of interest:

• `coliny_beta`
• `coliny_direct`
• `coliny_pattern_search`
• `coliny_ea`
• `coliny_solis_wets`

**initial_delta**

• Keywords Area
• method
• `coliny_cobyla`
• `initial_delta`

Reasonable initial changes to optimization variables

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 1.0 (COBYLA), 0.1*range (PS, SW)

**Description**

`initial_delta` for COBYLA should be set to be a value that is reasonable for initial changes to the optimization variables. It represents a distance from the initial simplex within which linear approximation subproblems can be trusted to be sufficiently representative of the true problem. It is analogous to the initial trust-region size used in trust-region methods.

**Default Behavior**

The default value is 1.0.

**threshold_delta**

• Keywords Area
• method
• `coliny_cobyla`
• `threshold_delta`

Required or expected accuracy in optimization variables.
6.2. **METHOD**

---

**Specification**

**Alias**: none  
**Argument(s)**: REAL  
**Default**: 1.0e-4 (COBYLA), 1.0e-5 (PS), 1.0e-6 (SW)

**Description**

Unlike it’s use as a stopping criteria in other methods, `threshold_delta` is used to communicate the accuracy of the optimization variables to COBYLA. It represents the minimum distance from the simplex within which linear approximation subproblems can be trusted to be sufficiently representative of the true problem. It is analogous to the minimum trust-region size used in trust-region methods. Note that, per COBYLA documentation, the level of accuracy is not guaranteed.

**Default Behavior**

The default value is 0.0001.

**Additional Discussion**

While `threshold_delta` is not a stopping criteria for COBYLA, we note that it may have a side effect regarding convergence. In particular, if lower accuracy is required of the optimization variables, it may stop sooner than if higher accuracy is required.

---

**solution_target**

- **Keywords Area**
- **method**
- **coliny_cobyla**
- **solution_target**

Stopping criteria based on objective function value

---

**Specification**

**Alias**: `solution_accuracy`  
**Argument(s)**: REAL  
**Default**: -DBL_MAX

**Description**

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

---

**seed**

- **Keywords Area**
- **method**
- **coliny_cobyla**
- **seed**

Seed of the random number generator
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

method
sampling
    sample_type lhs
    samples = 10
    seed = 15347

show_misc_options

• Keywords Area
• method
• coliny_cobyla
• show_misc_options

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

Argument(s): none

Default: no dump of specification options

Description

All SCOLIB methods support the show_misc_options optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The misc_options optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using
this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

**misc_options**

- **Keywords Area**
- **method**
- **coliny_cobyla**
- **misc_options**

Set method options not available through Dakota spec

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** no misc options

**Description**

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The **misc_options** optional specification provides a means for inputing additional settings supported by the SCOLIB-B methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

**max_iterations**

- **Keywords Area**
- **method**
- **coliny_cobyla**
- **max_iterations**

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**
**Specification**

Alias: none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25+n)

**Description**

The maximum number of iterations (default: 100). See also `max_function_evaluations`.

**convergence_tolerance**

- Keywords Area
- method
- coliny_cobyla
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

**Argument(s):** REAL

**Default:** 1.e-4

**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
6.2. METHOD

- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

- NL2SOL: See nl2sol

max_function_evaluations

- Keywords Area
- method
- coliny_cobyla
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
Argument(s): INTEGER
Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling

- Keywords Area
- method
- coliny_cobyla
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): none
  Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all scale types and ∗scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a ∗scales keyword, which gives characteristic values
- a ∗scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of ∗scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   - two-sided bounds scaled into the interval [0,1];
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. log - logarithmic scaling
   First, any characteristic values from the optional ∗scales specification are applied. Then logarithm base 10 scaling is applied.
   Logarithmic scaling is not available for linear constraints.
   When continuous design variables are log scaled, linear constraints are not allowed.
6.2. METHOD

Scales

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above. Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0\times10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

model_pointer

- Keywords Area
- method
- coliny_cobyla
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYWORDS AREA

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
   tabular_graphics_data
   method_pointer = 'UQ'

method
   id_method = 'UQ'
   model_pointer = 'Surr'
   sampling,
      samples = 10
      seed = 98765 rng rnum2
      response_levels = 0.1 0.2 0.6
         0.1 0.2 0.6
      0.1 0.2 0.6
   sample_type lhs
   distribution cumulative

model
   id_model = 'Surr'
   surrogate global,
   dace_method_pointer = 'DACE'
   polynomial quadratic

method
   id_method = 'DACE'
   model_pointer = 'DACE_M'
   sampling sample_type lhs
      samples = 121 seed = 5034 rng rnum2

model
   id_model = 'DACE_M'
   single
   interface_pointer = 'I1'

variables
   uniform_uncertain = 2
      lower_bounds = 0. 0.
      upper_bounds = 1. 1.
   descriptors = 'x1' 'x2'

interface
   id_interface = 'I1'
   system asynch evaluation_concurrency = 5
      analysis_driver = 'text_book'

responses
6.2. METHOD

response_functions = 3
no_gradients
no_hessians

6.2.37 coliny_direct

- Keywords Area
- method
- coliny_direct

DIviding RECTangles method

Topics
This keyword is related to the topics:

- package_scolib
- package_coliny
- global_optimization_methods

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>division</td>
<td>Determine how rectangles are subdivided</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>global_balance_-parameter</td>
<td>Tolerance for whether a subregion is worth dividing</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>local_balance_-parameter</td>
<td>Tolerance for whether a subregion is worth dividing</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_boxsize_limit</td>
<td>Stopping Criterion based on longest edge of hyperrectangle</td>
</tr>
<tr>
<td>Optional</td>
<td>min_boxsize_limit</td>
<td>Stopping Criterion based on shortest edge of hyperrectangle</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>constraint_penalty</td>
<td>Multiplier for the penalty function</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>solution_target</td>
<td>Stopping criteria based on objective function value</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>Seed of the random number generator</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>show_misic_options</td>
<td>Show algorithm parameters not exposed in Dakota input</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>misc_options</td>
<td>Set method options not available through Dakota spec</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The DIViding RECTangles (DIRECT) optimization algorithm is a derivative free global optimization method that balances local search in promising regions of the design space with global search in unexplored regions. As shown in Figure 5.1, DIRECT adaptively subdivides the space of feasible design points so as to guarantee that iterates are generated in the neighborhood of a global minimum in finitely many iterations.

DIRECT” \image latex direct1.eps ”Design space partitioning with DIRECT” width=10cm
6.2. METHOD

In practice, DIRECT has proven an effective heuristic for engineering design applications, for which it is able to quickly identify candidate solutions that can be further refined with fast local optimizers.

See the page package_scolib for important information regarding all SCOLIB methods
The DIRECT algorithm supports concurrency up to twice the number of variables being optimized.
DIRECT uses the solution_target, constraint_penalty and show_misc_options specifications that are described in package_scolib. Note, however, that DIRECT uses a fixed penalty value for constraint violations (i.e. it is not dynamically adapted as is done in coliny_pattern_search).

Search Parameters
The global_balance_parameter controls how much global search is performed by only allowing a subregion to be subdivided if the size of the subregion divided by the size of the largest subregion is at least global_balance_parameter. Intuitively, this forces large subregions to be subdivided before the smallest subregions are refined. The local_balance_parameter provides a tolerance for estimating whether the smallest subregion can provide a sufficient decrease to be worth subdividing; the default value is a small value that is suitable for most applications.

Stopping Criteria
DIRECT can be terminated with:
- max_function_evaluations
- max_iterations
- convergence_tolerance
- solution_target
- max_boxsize_limit
- min_boxsize_limit - most effective in practice

See Also
These keywords may also be of interest:
- coliny_beta
- coliny_pattern_search
- coliny_cobyla
- coliny_ea
- coliny_solis_wets
- ncsu_direct

division

- Keywords Area
- method
- coliny_direct
- division

Determine how rectangles are subdivided
**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** major\_dimension

<table>
<thead>
<tr>
<th>Required/-Optional Required</th>
<th>Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose One</td>
<td></td>
<td>major_dimension</td>
<td>(default) Longest edge of subregion is subdivided</td>
</tr>
<tr>
<td>all_dimensions</td>
<td></td>
<td>all_dimensions</td>
<td>All dimensions are simultaneously subdivided</td>
</tr>
</tbody>
</table>

**Description**

The `division` specification determines how DIRECT subdivides each subregion of the search space.

If `division` is set to `major\_dimension`, then the dimension representing the longest edge of the subregion is subdivided (this is the default). If `division` is set to `all\_dimensions`, then all dimensions are simultaneously subdivided.

**major\_dimension**

- Keywords Area
- method
- coliny\_direct
- division
- major\_dimension

  (default) Longest edge of subregion is subdivided

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

Longest edge of subregion is subdivided

**all\_dimensions**

- Keywords Area
- method
- coliny\_direct
- division
- all\_dimensions

All dimensions are simultaneously subdivided
6.2. METHOD

Specification

Alias: none
  Argument(s): none

Description

All dimensions are simultaneously subdivided

global_balance_parameter

- Keywords Area
- method
- coliny_direct
- global_balance_parameter

Tolerance for whether a subregion is worth dividing

Specification

Alias: none
  Argument(s): REAL
  Default: 0.0

Description

The global_balance_parameter controls how much global search is performed by only allowing a subregion to be subdivided if the size of the subregion divided by the size of the largest subregion is at least global_balance_parameter. Intuitively, this forces large subregions to be subdivided before the smallest subregions are refined.

local_balance_parameter

- Keywords Area
- method
- coliny_direct
- local_balance_parameter

Tolerance for whether a subregion is worth dividing

Specification

Alias: none
  Argument(s): REAL
  Default: 1.e-8
CHAPTER 6. KEYWORDS AREA

Description

See parent page. The local_balance_parameter provides a tolerance for estimating whether the smallest subregion can provide a sufficient decrease to be worth subdividing; the default value is a small value that is suitable for most applications.

max_boxsize_limit

- Keywords Area
- method
- coliny_direct
- max_boxsize_limit

Stopping Criterion based on longest edge of hyperrectangle

Specification

Alias: none
Argument(s): REAL
Default: 0.0

Description

Each subregion considered by DIRECT has a size, which corresponds to the longest diagonal of the subregion. max_boxsize_limit specification terminates DIRECT if the size of the largest subregion falls below this threshold.

min_boxsize_limit

- Keywords Area
- method
- coliny_direct
- min_boxsize_limit

Stopping Criterion based on shortest edge of hyperrectangle

Specification

Alias: none
Argument(s): REAL
Default: 1.0e-4
Description

\textit{min\_boxsize\_limit} is a setting that terminates the optimization when the measure of a hyperrectangle \( S \) with \( f(c(S)) = \text{fmin} \) is less than \( \text{min\_boxsize\_limit} \).

Each subregion considered by DIRECT has a \textit{size}, which corresponds to the longest diagonal of the subregion. \textit{min\_boxsize\_limit} specification terminates DIRECT if the size of the smallest subregion falls below this threshold.

In practice, this specification is likely to be more effective at limiting DIRECT’s search.

\textbf{constraint\_penalty}

- \textit{Keywords Area}
- \textit{method}
- \textit{coliny\_direct}
- \textit{constraint\_penalty}

Multiplier for the penalty function

\textbf{Specification}

\textbf{Alias}: none

\textbf{Argument(s)}: REAL

\textbf{Default}: 1000.0

\textbf{Description}

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds \textit{constraint\_penalty} times the sum of squares of the constraint violations to the objective function. The default value of \textit{constraint\_penalty} is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

\textbf{solution\_target}

- \textit{Keywords Area}
- \textit{method}
- \textit{coliny\_direct}
- \textit{solution\_target}

Stopping criteria based on objective function value

\textbf{Specification}

\textbf{Alias}: solution\_accuracy

\textbf{Argument(s)}: REAL

\textbf{Default}: \(-\text{DBL\_MAX}\)
CHAPTER 6. KEYWORDS AREA

Description

solution_target is a termination criterion. The algorithm will terminate when the function value falls below solution_target.

seed

- Keywords Area
- method
- coliny_direct
- seed

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

method
    sampling
        sample_type lhs
        samples = 10
        seed = 15347

show_m misc.options

- Keywords Area
- method
- coliny_direct
- show_m misc.options

Show algorithm parameters not exposed in Dakota input
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: no dump of specification options

Description

All SCOLIB methods support the show_misc_options optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The misc_options optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

misc_options

- Keywords Area
- method
- coliny_direct
- misc_options

Set method options not available through Dakota spec

Specification

Alias: none
Argument(s): STRINGLIST
Default: no misc options

Description

All SCOLIB methods support the show_misc_options optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The misc_options optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

max_iterations

- Keywords Area
- method
- coliny_direct
- **max_iterations**

  Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

Alias: none

- **Argument(s):** INTEGER
  - **Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence}/efficient-global: 25+\( n \))

**Description**

The maximum number of iterations (default: 100). See also `max_function_evaluations`.

**convergence_tolerance**

- **Keywords Area**

- **method**

- **coliny_direct**

- **convergence_tolerance**

  Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

Alias: none

- **Argument(s):** REAL
  - **Default:** 1.e-4
6.2. METHOD

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_function_evaluations

- **Keywords Area**
- **method**
- **coliny_direct**
- **max_function_evaluations**

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

**Alias**: none

**Argument(s)**: INTEGER

**Default**: 1000

Description

The maximum number of function evaluations (default: 1000). See also `max_iterations`. 
CHAPTER 6. KEYWORDS AREA

scaling

- Keywords Area
- method
- coliny_direct
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. **auto** - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval \([0,1]\);
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into \([0,1]\).

4. **log** - logarithmic scaling

First, any characteristic values from the optional \(*_{\text{scales}}\) specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The \(*_{\text{scales}}\) keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding \(*_{\text{scale_type}}\), as described above.

Depending on the scale type, the characteristic values may be required or optional.

- **none, auto, log** - optional
- **value** - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0\times10^{\text{DBL_MIN}}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
model_pointer

- Keywords Area
- method
- coliny_direct
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
tabular_graphics_data
method_pointer = ’UQ’

method
id_method = ’UQ’
model_pointer = ’SURR’
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

distribution
model
id_model = ’SURR’
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.38 coliny_ea

- Keywords Area
- method
- coliny_ea

Evolutionary Algorithm

Topics
This keyword is related to the topics:

- package_scolib
- package_coliny
- global_optimization_methods

Specification
Alias: none
Argument(s): none
<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>population size</td>
<td>population_size</td>
<td>Set the population size</td>
</tr>
<tr>
<td>Optional</td>
<td>initialization type</td>
<td>initialization_type</td>
<td>Specify how to initialize the population</td>
</tr>
<tr>
<td>Optional</td>
<td>fitness type</td>
<td>fitness_type</td>
<td>Select fitness type</td>
</tr>
<tr>
<td>Optional</td>
<td>replacement type</td>
<td>replacement_type</td>
<td>Select a replacement type for SCOLIB evolutionary algorithm (coliny_earth)</td>
</tr>
<tr>
<td>Optional</td>
<td>crossover rate</td>
<td>crossover_rate</td>
<td>Specify the probability of a crossover event</td>
</tr>
<tr>
<td>Optional</td>
<td>crossover type</td>
<td>crossover_type</td>
<td>Select a crossover type</td>
</tr>
<tr>
<td>Optional</td>
<td>mutation rate</td>
<td>mutation_rate</td>
<td>Set probability of a mutation</td>
</tr>
<tr>
<td>Optional</td>
<td>mutation type</td>
<td>mutation_type</td>
<td>Select a mutation type</td>
</tr>
<tr>
<td>Optional</td>
<td>constraint penalty</td>
<td>constraint_penalty</td>
<td>Multiplier for the penalty function</td>
</tr>
<tr>
<td>Optional</td>
<td>solution target</td>
<td>solution_target</td>
<td>Stopping criteria based on objective function value</td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td>show_misec_options</td>
<td>show_misec_options</td>
<td>Show algorithm parameters not exposed in Dakota input</td>
</tr>
<tr>
<td>Optional</td>
<td>misc_options</td>
<td>misc_options</td>
<td>Set method options not available through Dakota spec</td>
</tr>
</tbody>
</table>
### Description

Evolutionary Algorithm

See the page `package_scolib` for important information regarding all SCOLIB methods

`coliny_pattern_search` supports concurrency up to the size of the population

The random seed control provides a mechanism for making a stochastic optimization repeatable. That is, the use of the same random seed in identical studies will generate identical results. The `population_size` control specifies how many individuals will comprise the EA’s population.

The `initialization_type` defines the type of initialization for the population of the EA. There are three types: `simple_random`, `unique_random`, and `flat_file`. `simple_random` creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs. The number of designs is specified by the `population_size`. `unique_random` is the same as `simple_random`, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected. `flat_file` allows the initial population to be read from a flat file. If `flat_file` is specified, a file name must be given.

The `fitness_type` controls how strongly differences in "fitness" (i.e., the objective function) are weighted in the process of selecting "parents" for crossover:

- the `linear_rank` setting uses a linear scaling of probability of selection based on the rank order of each individual’s objective function within the population

- the `merit_function` setting uses a proportional scaling of probability of selection based on the relative value of each individual’s objective function within the population

The `replacement_type` controls how current populations and newly generated individuals are combined to create a new population. Each of the `replacement_type` selections accepts an integer value, which is referred to below as the `replacement_size`.

<table>
<thead>
<tr>
<th>Optional</th>
<th>max_iterations</th>
<th>Stopping criterion based on number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
<tr>
<td>Optional</td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>
• The random setting creates a new population using (a) replacement_size randomly selected individuals from the current population, and (b) population_size - replacement_size individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using new_solutions_generated) that are created for each generation (using the selection, crossover, and mutation procedures).

• The chc setting creates a new population using (a) the replacement_size best individuals from the combination of the current population and the newly generated individuals, and (b) population_size - replacement_size individuals randomly selected from among the remaining individuals in this combined pool. The chc setting is the preferred selection for many engineering problems.

• The elitist (default) setting creates a new population using (a) the replacement_size best individuals from the current population, (b) and population_size - replacement_size individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default new_solutions_generated value is set such that the entire set of newly generated individuals will be selected for replacement.

Note that new_solutions_generated is not recognized by Dakota as a valid keyword unless replacement_type has been specified.

**Theory**

The basic steps of an evolutionary algorithm are depicted in Figure 5.2.

![Figure 6.2: Depiction of evolutionary algorithm](image.png)

They can be enumerated as follows:

1. Select an initial population randomly and perform function evaluations on these individuals
2. Perform selection for parents based on relative fitness
3. Apply crossover and mutation to generate new solutions generated new individuals from the selected parents
   - Apply crossover with a fixed probability from two selected parents
   - If crossover is applied, apply mutation to the newly generated individual with a fixed probability
   - If crossover is not applied, apply mutation with a fixed probability to a single selected parent

4. Perform function evaluations on the new individuals

5. Perform replacement to determine the new population

6. Return to step 2 and continue the algorithm until convergence criteria are satisfied or iteration limits are exceeded

See Also

These keywords may also be of interest:
- coliny_beta
- coliny_direct
- coliny_pattern_search
- coliny_cobyla
- coliny_solis_wets

population_size
- Keywords Area
- method
- coliny_ea
- population_size

Set the population size

Specification

Alias: none

Argument(s): INTEGER

Default: 50

Description

The number of designs in the population is specified by the population_size.
**initialization_type**

- **Keywords Area**
- **method**
- **coliny_ea**
- **initialization_type**

Specify how to initialize the population

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** unique_random

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple_random</td>
<td>Create random initial solutions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unique_random</td>
<td>Create random initial solutions, but enforce uniqueness (default)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flat_file</td>
<td>Read initial solutions from file</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The `initialization_type` defines how the initial population is created for the GA. There are three types:

1. `simple_random`
2. `unique_random` (default)
3. `flat_file`

Setting the size for the `flat_file` initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the `unique_random` initializer and then the `simple_random` initializer if necessary.

**simple_random**

- **Keywords Area**
- **method**
- **coliny_ea**
- **initialization_type**
- **simple_random**

Create random initial solutions
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

simple.random creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs.

unique.random

- Keywords Area
- method
- coliny_ea
- initialization_type
- unique_random

Create random initial solutions, but enforce uniqueness (default)

Specification

Alias: none
Argument(s): none

Description

unique.random is the same as simple.random, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected.

flat_file

- Keywords Area
- method
- coliny_ea
- initialization_type
- flat_file

Read initial solutions from file

Specification

Alias: none
Argument(s): STRING
**Description**

`flat_file` allows the initial population to be read from a flat file. If `flat_file` is specified, a file name must be given.

**fitness_type**

- Keywords Area
- method
- coliny_ea
- fitness_type

Select fitness type

**Specification**

Alias: none

Argument(s): none

Default: linear_rank

<table>
<thead>
<tr>
<th>Required/-Optional Required</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(Choose One)</em></td>
<td></td>
<td>linear_rank</td>
<td>Set selection scaling</td>
</tr>
<tr>
<td>merit_function</td>
<td></td>
<td></td>
<td>Balance goals of reducing objective function and satisfying constraints</td>
</tr>
</tbody>
</table>

**Description**

The `fitness_type` controls how strongly differences in "fitness" (i.e., the objective function) are weighted in the process of selecting "parents" for crossover. It has two options, `linear_rank` and `merit_function`.

**linear_rank**

- Keywords Area
- method
- coliny_ea
- fitness_type
- linear_rank

Set selection scaling

**Specification**

Alias: none

Argument(s): none
6.2. METHOD

Description
The linear_rank setting uses a linear scaling of probability of selection based on the rank order of each individual’s objective function within the population.

merit_function

- Keywords Area
- method
- coliny_ea
- fitness_type
- merit_function

Balance goals of reducing objective function and satisfying constraints.

Specification
Alias: none
Argument(s): none

Description
A merit_function is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

replacement_type

- Keywords Area
- method
- coliny_ea
- replacement_type

Select a replacement type for SCOLIB evolutionary algorithm (coliny_ea).

Specification
Alias: none
Argument(s): none
Default: elitist=1

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group 1</td>
<td>random</td>
<td>Description</td>
</tr>
<tr>
<td>Required*</td>
<td>(Choose One)</td>
<td></td>
<td>Create new population randomly</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>chc</th>
<th>Create new population using replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>elitist</td>
<td>Use the best designs to form a new population</td>
</tr>
</tbody>
</table>

Optional

| new_solutions_generated | Replace population with individuals chosen from population |

**Description**

The replacement_type controls how current populations and newly generated individuals are combined to create a new population. Each of the replacement_type selections accepts an associated integer value, which is specified by the replacement_size:

The random setting creates a new population using (a) replacement_size randomly selected individuals from the current population, and (b) population_size - replacement_size individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using new_solutions_generated) that are created for each generation (using the selection, crossover, and mutation procedures).

The chc setting creates a new population using (a) the replacement_size best individuals from the combination of the current population and the newly generated individuals, and (b) population_size - replacement_size individuals randomly selected from among the remaining individuals in this combined pool. The chc setting is the preferred selection for many engineering problems.

The elitist (default) setting creates a new population using (a) the replacement_size best individuals from the current population, (b) and population_size - replacement_size individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default new_solutions_generated value is set such that the entire set of newly generated individuals will be selected for replacement.

Note that new_solutions_generated is not recognized by Dakota as a valid keyword unless replacement_type has been specified.

**Specification**

* Alias: none
  * Argument(s): INTEGER
**6.2. METHOD**

**Description**

The replacement_type controls how current populations and newly generated individuals are combined to create a new population. Each of the replacement_type selections accepts an integer value, which is referred as the replacement_size:

The random setting creates a new population using:

- replacement_size randomly selected individuals from the current population, and
- population_size - replacement_size individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using new_solutions_generated) that are created for each generation (using the selection, crossover, and mutation procedures).

**chc**

- Keywords Area
- method
- coliny_ea
- replacement_type
- chc

Create new population using replacement

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

The replacement_type controls how current populations and newly generated individuals are combined to create a new population. Each of the replacement_type selections accepts an integer value, which is referred as the replacement_size:

The chc setting creates a new population using (a) the replacement_size best individuals from the combination of the current population and the newly generated individuals, and (b) population_size - replacement_size individuals randomly selected from among the remaining individuals in this combined pool. The chc setting is the preferred selection for many engineering problems.

**elitist**

- Keywords Area
- method
- coliny_ea
- replacement_type
- elitist

Use the best designs to form a new population
**Specification**

Alias: none  
**Argument(s):** INTEGER

**Description**

The *elitist* (default) setting creates a new population using (a) the `replacement_size` best individuals from the current population, (b) and `population_size - replacement_size` individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default `new_solutions_generated` value is set such that the entire set of newly generated individuals will be selected for replacement.

`new_solutions_generated`

- Keywords Area
- method
- coliny_ea
- replacement_type
- `new_solutions_generated`

Replace population with individuals chosen from population

**Specification**

Alias: none  
**Argument(s):** INTEGER  
**Default:** `population_size - replacement_size`

**Description**

- The *random* setting creates a new population using (a) `replacement_size` randomly selected individuals from the current population, and (b) `population_size - replacement_size` individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using `new_solutions_generated`) that are created for each generation (using the selection, crossover, and mutation procedures).

`crossover_rate`

- Keywords Area
- method
- coliny_ea
- `crossover_rate`

Specify the probability of a crossover event
6.2. **METHOD**

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** 0.8

**Description**

The crossover type controls what approach is employed for combining parent genetic information to create offspring, and the crossover rate specifies the probability of a crossover operation being performed to generate a new offspring. The SCOLIB EA method supports three forms of crossover, two point, blend, and uniform, which generate a new individual through combinations of two parent individuals. Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate. Uniform crossover creates offspring through random combination of coordinates from the two parents. Blend crossover generates a new individual randomly along the multidimensional vector connecting the two parents.

**crossover_type**

- **Keywords Area**
- **method**
- **coliny_ea**
- **crossover_type**

Select a crossover type

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> (Choose One)</td>
<td><strong>two_point</strong></td>
<td><strong>two_point</strong></td>
<td>Combine middle of one parent with end of another</td>
</tr>
</tbody>
</table>

| | | | |
| blend | Random blend of parents |
| uniform | Randomly combine coordinates from parents |

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** two_point

**Description**

The crossover type controls what approach is employed for combining parent genetic information to create offspring. The SCOLIB EA method supports three forms of crossover, two point, blend, and uniform, which generate a new individual through combinations of two parent individuals.
two_point

- Keywords Area
- method
- coliny_ea
- crossover_type
- two_point

Combine middle of one parent with end of another

**Specification**

Alias: none

Argument(s): none

**Description**

Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate.

blend

- Keywords Area
- method
- coliny_ea
- crossover_type
- blend

Random blend of parents

**Specification**

Alias: none

Argument(s): none

**Description**

blend crossover generates a new individual randomly along the multidimensional vector connecting the two parents.
6.2. METHOD

uniform
• Keywords Area
• method
• coliny_ea
• crossover_type
• uniform
  Randomly combine coordinates from parents

Specification
Alias: none
Argument(s): none

Description
Uniform crossover creates offspring through random combination of coordinates from the two parents.

mutation_rate
• Keywords Area
• method
• coliny_ea
• mutation_rate
  Set probability of a mutation

Specification
Alias: none
Argument(s): REAL
Default: 1.0

Description
The mutation_rate controls the probability of mutation being performed on an individual, both for new individuals generated by crossover (if crossover occurs) and for individuals from the existing population. It is the fraction of trial points that are mutated in a given iteration and therefore must be specified to be between 0 and 1.

mutation_type
• Keywords Area
• method
• coliny_ea
• mutation_type
  Select a mutation type
### Specification

**Alias:** none

**Argument(s):** none

**Default:** offset_normal

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required (Choose One)</strong></td>
<td></td>
<td>replace_uniform</td>
<td>Replace coordinate with randomly generated value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>offset_normal</td>
<td>Set mutation offset to use a normal distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>offset_cauchy</td>
<td>Use a Cauchy distribution for the mutation offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>offset_uniform</td>
<td>Set mutation offset to use a uniform distribution</td>
</tr>
</tbody>
</table>

| Optional | | non_adaptive | Disable self-adaptive mutation |

### Description

The `mutation_type` controls what approach is employed in randomly modifying continuous design variables within the EA population. Each of the mutation methods generates coordinate-wise changes to individuals, usually by adding a random variable to a given coordinate value (an `offset_*` mutation), but also by replacing a given coordinate value with a random variable (a `replace_*` mutation).

Discrete design variables are always mutated using the `offset_uniform` method.

#### replace_uniform

- Keywords Area
- method
- `coliny_ea`
- `mutation_type`
- `replace_uniform`

Replace coordinate with randomly generated value

### Specification

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description

The replace_uniform mutation type generates a replacement value for a coordinate using a uniformly distributed value over the total range for that coordinate.

offset_normal

- Keywords Area
- method
- coliny_ea
- mutation_type
- offset_normal

Set mutation offset to use a normal distribution

Specification

Alias: none
Argument(s): none

| Required/- | Description of | Dakota Keyword | Dakota Keyword | Description |
| Optional   | Group          |               |               |            |
| Optional   | mutation_scale | mutation_scale | mutation_range | Scales mutation across range of parameter |
|            |                |                |                | Set uniform offset control for discrete parameters |

Description

The offset_normal type is an "offset" mutation that adds a 0-mean random variable with a normal uniform distribution to the existing coordinate value. The offset is limited in magnitude by mutation_scale.

mutation_scale

- Keywords Area
- method
- coliny_ea
- mutation_type
- offset_normal
- mutation_scale

Scales mutation across range of parameter
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REAL
Default: 0.1

Description

The `mutation_scale` specifies a scale factor which scales continuous mutation offsets; this is a fraction of the total range of each dimension, so `mutation_scale` is a relative value between 0 and 1.

```
mutation_range
```

- Keywords Area
- method
- coliny_ea
- mutation_type
- offset_normal
- mutation_range

Set uniform offset control for discrete parameters

Specification

Alias: none

Argument(s): INTEGER
Default: 1

Description

The `mutation_range` is used to control `offset_uniform` mutation used for discrete parameters. The replacement discrete value is the original value plus or minus an integer value up to `mutation_range`.

```
offset_cauchy
```

- Keywords Area
- method
- coliny_ea
- mutation_type
- offset_cauchy

Use a Cauchy distribution for the mutation offset

Specification

Alias: none

Argument(s): none
Description

The `offset_cauchy` type is an “offset” mutation that adds a 0-mean random variable with a cauchy distribution to the existing coordinate value. The offset is limited in magnitude by `mutation_scale`.

```
mutation_scale
  • Keywords Area
  • method
  • coliny_ea
  • mutation_type
  • offset_cauchy
  • mutation_scale
```

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL
Default: 0.1

Description

The `mutation_scale` specifies a scale factor which scales continuous mutation offsets; this is a fraction of the total range of each dimension, so `mutation_scale` is a relative value between 0 and 1.

```
motion_range
  • Keywords Area
  • method
  • coliny_ea
  • mutation_type
  • offset_cauchy
  • mutation_range
```

Set uniform offset control for discrete parameters
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
Argument(s): INTEGER
Default: 1

Description
The mutation_range is used to control offset_uniform mutation used for discrete parameters. The replacement discrete value is the original value plus or minus an integer value up to mutation_range.

offset_uniform
- Keywords Area
- method
- coliny_ea
- mutation_type
- offset_uniform

Set mutation offset to use a uniform distribution

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>mutation_scale</td>
<td>Scales mutation across range of parameter</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>mutation_range</td>
<td>Set uniform offset control for discrete parameters</td>
</tr>
</tbody>
</table>

Description
The offset_uniform type is an "offset" mutation that adds a 0-mean random variable with a uniform distribution to the existing coordinate value. The offset is limited in magnitude by mutation_scale.

For discrete design variables, offset_uniform is always used, and mutation_range controls the magnitude of the mutation.

mutation_scale
- Keywords Area
- method
- coliny_ea
- mutation_type
6.2. **METHOD**

- **offset_uniform**
- **mutation_scale**

Scales mutation across range of parameter

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 0.1

**Description**

The `mutation_scale` specifies a scale factor which scales continuous mutation offsets; this is a fraction of the total range of each dimension, so `mutation_scale` is a relative value between 0 and 1.

- **mutation_range**
  - **Keywords Area**
  - **method**
  - **coliny_ea**
  - **mutation_type**
  - **offset_uniform**
  - **mutation_range**

Set uniform offset control for discrete parameters

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 1

**Description**

The `mutation_range` is used to control `offset_uniform` mutation used for discrete parameters. The replacement discrete value is the original value plus or minus an integer value up to `mutation_range`.

- **nonadaptive**
  - **Keywords Area**
  - **method**
  - **coliny_ea**
  - **mutation_type**
  - **nonadaptive**

Disable self-adaptive mutation
CHAPTER 6. KEYWORDS AREA

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** Adaptive mutation

**Description**

The SCOLIB EA method uses self-adaptive mutation, which modifies the mutation scale dynamically. This mechanism is borrowed from EAs like evolution strategies. The `non_adaptive` flag can be used to deactivate the self-adaptation, which may facilitate a more global search.

Note that `non_adaptive` is not recognized by Dakota as a valid keyword unless `mutation_type` has been specified.

**constraint_penalty**

- **Keywords Area**
  - **method**
  - **coliny_ea**
  - **constraint_penalty**

  Multiplier for the penalty function

**Specification**

**Alias:** none  
**Argument(s):** REAL

**Description**

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

**solution_target**

- **Keywords Area**
  - **method**
  - **coliny_ea**
  - **solution_target**

  Stopping criteria based on objective function value
6.2. METHOD

Specification

Alias: solution_accuracy
Argument(s): REAL
Default: -DBL_MAX

Description

solution_target is a termination criterion. The algorithm will terminate when the function value falls below solution_target.

seed

- Keywords Area
- method
- coliny_ea
- seed

Seed of the random number generator

Specification

Alias: none
Argument(s): INTEGER
Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

method sampling
  sample_type lhs
  samples = 10
  seed = 15347
show_m misc_options

- Keywords Area
- method
- coliny_ea
- show_m misc_options

Show algorithm parameters not exposed in Dakota input

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no dump of specification options

**Description**

All SCOLIB methods support the `show_m misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

misc_options

- Keywords Area
- method
- coliny_ea
- misc_options

Set method options not available through Dakota spec

**Specification**

**Alias:** none  
**Argument(s):** STRINGLIST  
**Default:** no misc options
6.2. METHOD

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

`max_iterations`

- **Keywords Area**
- **method**
- **coliny_ea**
- **max_iterations**

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- **method_independent_controls**

Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: `fsu_cvt`, `local_reliability`: 25; `global_\{reliability , interval_est , evidence\}` / `efficient_global`: 25*n)

Description

The maximum number of iterations (default: 100). See also `max_function_evaluations`.

`convergence_tolerance`

- **Keywords Area**
- **method**
- **coliny_ea**
- **convergence_tolerance**

Stopping criterion based on convergence of the objective function or statistics
 Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: 1.e-4

Description
The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration. Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library
This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_function_evaluations

- Keywords Area
- method
- coliny_ea
- max_function_evaluations

Stopping criteria based on number of function evaluations
6.2. METHOD

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max_itera\tions.

scaling
- Keywords Area
- method
- coliny_ea
- scaling
Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.
• a *scales keyword, which gives characteristic values

• a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored

2. value - multiplicative scaling

3. auto - automatic scaling

   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

   • two-sided bounds scaled into the interval [0,1];
   • one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   • no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling

   First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

   Logarithmic scaling is not available for linear constraints.

   When continuous design variables are log scaled, linear constraints are not allowed.

Scales
The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

• none, auto, log - optional

• value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
Examples

The two examples below are equivalent:

```
responses
    objective_functions 3
    sense "maximize"
    primary_scale_types = "value"
    primary_scales = 1 1 100

responses
    objective_functions 3
    sense "maximize"
    primary_scale_types = "value" "value" "value"
    primary_scales = 1 1 100
```

**model_pointer**

- Keywords Area
- method
- coliny_ea
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.
Examples

environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.39  coliny_beta

•  Keywords Area

•  method

•  coliny_beta

(Experimental) Coliny beta solver
6.2. METHOD

Topics
This keyword is related to the topics:

- package_scolib
- package_coliny

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>beta_solver_name</td>
<td>Use an in-development SCOLIB solver</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>solution_target</td>
<td>Stopping criteria based on objective function value</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>show_m misc_options</td>
<td>Show algorithm parameters not exposed in Dakota input</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>misc_options</td>
<td>Set method options not available through Dakota spec</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_-,tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
</tbody>
</table>
### Description

This method keyword allows testing of experimental (beta) Coliny (Solib) optimization solvers during software development. It is intended primarily for developer use. Additional information on Coliny solvers is available at [package_solib].

### See Also

These keywords may also be of interest:
- `coliny_direct`
- `coliny_pattern_search`
- `coliny_cobyla`
- `coliny_ea`
- `coliny_solis_wets`

**beta_solver_name**

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [beta_solver_name](#)

Use an in-development SCOLIB solver

### Specification

**Alias**: none

**Argument(s)**: STRING

### Description

This is a means of accessing new methods in SCOLIB before they are exposed through the Dakota interface. Seek help from a Dakota or SCOLIB developer or a Dakota developer.
6.2. METHOD

**solution_target**

- Keywords Area
- method
- coliny_beta
- solution_target

Stopping criteria based on objective function value

**Specification**

*Alias*: solution_accuracy  
*Argument(s)*: REAL  
*Default*: -DBL_MAX

**Description**

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

**seed**

- Keywords Area
- method
- coliny_beta
- seed

Seed of the random number generator

**Specification**

*Alias*: none  
*Argument(s)*: INTEGER  
*Default*: system-generated (non-repeatable)

**Description**

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.
Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

show_misc_options

- Keywords Area
- method
- coliny_beta
- show_misc_options

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

- Argument(s): none
- Default: no dump of specification options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

misc_options

- Keywords Area
- method
- coliny_beta
- misc_options

Set method options not available through Dakota spec

Specification

Alias: none

- Argument(s): STRINGLIST
- Default: no misc options
6.2. METHOD

Description

All SCOLIB methods support the show_mis_opts optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The mis_opts optional specification provides a means for inputing additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

max_iterations

- Keywords Area
- method
- coliny_beta
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Defaut: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- coliny_beta
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics
Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

- Argument(s): REAL
- Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_function_evaluations

- Keywords Area
- method
- coliny_beta
- max_function_evaluations

Stopping criteria based on number of function evaluations
6.2. **METHOD**

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

Argument(s): INTEGER

Default: 1000

**Description**

The maximum number of function evaluations (default: 1000). See also max_iterations.

**scaling**

- Keywords Area
- method
- coliny_beta
- scaling

Turn on scaling for variables, responses, and constraints

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

Alias: none

Argument(s): none

Default: no scaling

**Description**

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and *_scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.
• a *scales keyword, which gives characteristic values

• a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types
There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored

2. value - multiplicative scaling

3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

• two-sided bounds scaled into the interval [0,1];
• one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
• no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales
The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

• none, auto, log - optional

• value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than 1.0e10*DBL_MIN. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
6.2. METHOD

Examples
The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

model_pointer

- Keywords Area
- method
- coliny_beta
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.
See block_pointer for details about pointers.
Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
      samples = 10
      seed = 98765 rng rnum2
      response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
      0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
      dace_method_pointer = 'DACE'
      polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
      samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
      lower_bounds = 0. 0.
      upper_bounds = 1. 1.
      descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
      analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.40 nl2sol

- Keywords Area
- method
- nl2sol

Trust-region method for nonlinear least squares
6.2. METHOD

Topics
This keyword is related to the topics:

- nonlinear_least_squares

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>function_precision</td>
<td>Specify the maximum precision of the analysis code responses</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>absolute_conv_tol</td>
<td>Absolute convergence tolerance</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>x_conv_tol</td>
<td>X-convergence tolerance</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>singular_conv_tol</td>
<td>Singular convergence tolerance</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>singular_radius</td>
<td>Singular radius</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>false_conv_tol</td>
<td>False convergence tolerance</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>initial_trust_radius</td>
<td>Initial trust region radius</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>covariance</td>
<td>Determine how the final covariance matrix is computed</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>regression_diagnostics</td>
<td>Turn on regression diagnostics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
</tbody>
</table>
NL2SOL is available as nl2sol and addresses unconstrained and bound-constrained least squares problems. It uses a trust-region method (and thus can be viewed as a generalization of the Levenberg-Marquardt algorithm) and adaptively chooses between two Hessian approximations, the Gauss-Newton approximation alone and the Gauss-Newton approximation plus a quasi-Newton approximation to the rest of the Hessian. Even on small-residual problems, the latter Hessian approximation can be useful when the starting guess is far from the solution. On problems that are not over-parameterized (i.e., that do not involve more optimization variables than the data support), NL2SOL usually exhibits fast convergence.

Several internal NL2SOL convergence tolerances are adjusted in response to function precision, which gives the relative precision to which responses are computed. These tolerances may also be specified explicitly using:

- convergence_tolerance (NL2SOL's rfctol)
- x_conv_tol (NL2SOL's xctol)
- absolute_conv_tol (NL2SOL's afctol)
- singular_conv_tol (NL2SOL's scctol)
- false_conv_tol (NL2SOL's xftol)
- initial_trust_radius (NL2SOL's lmax0)

The internal NL2SOL defaults can be obtained for many of these controls by specifying the value -1. The internal defaults are often functions of machine epsilon (as limited by function precision).

Examples

An example of nl2sol is given below, and is discussed in the User's Manual.

Note that in this usage of calibration terms, the driver script rosenbrock, is returning "residuals", which the nl2sol method is attempting to minimize. Another use case is to provide a data file, which Dakota will attempt
to match the model responses to. See calibration_data_file. Finally, as of Dakota 6.2, the field data capability may be used with nl2sol. That is, the user can specify field simulation data and field experiment data, and Dakota will interpolate and provide the proper residuals for the calibration.

```plaintext
# Dakota Input File: rosen_opt_nls.in
environment
tabular_data
tabular_data_file = 'rosen_opt_nls.dat'

method
  max_iterations = 100
  convergence_tolerance = 1e-4

model
  single
  variables
    continuous_design = 2
    initial_point -1.2 1.0
    lower_bounds -2.0 -2.0
    upper_bounds 2.0 2.0
    descriptors 'x1' 'x2'

interface
  analysis_driver = 'rosenbrock'
  direct

responses
  calibration_terms = 2
  analytic_gradients
  no_hessians
```

**Theory**

NL2SOL has a variety of internal controls as described in AT&T Bell Labs CS TR 153 (http://cm.bell-labs.com/cm/cs/cstr/153.ps.gz). A number of existing Dakota controls (method independent controls and responses controls) are mapped into these NL2SOL internal controls. In particular, Dakota’s convergence_tolerance, max_iterations, max_function_evaluations, and fd_gradient_step_size are mapped directly into NL2SOL’s rfctol, mxiter, mxfcal, and dltfdj controls, respectively. In addition, Dakota’s fd_hessian_step_size is mapped into both delta0 and dltdc, and Dakota’s output verbosity is mapped into NL2SOL’s auxprt and outlev (for normal/verbose/debug output, N- L2SOL prints initial guess, final solution, solution statistics, nondefault values, and changes to the active bound constraint set on every iteration; for quiet output, NL2SOL prints only the initial guess and final solution; and for silent output, NL2SOL output is suppressed).

**See Also**

These keywords may also be of interest:

- nlssol_sqp
- optpp_g_newton
- field_calibration_terms
function_precision

- Keywords Area
- method
- nl2sol
- function_precision

Specify the maximum precision of the analysis code responses

**Specification**

Alias: none

- **Argument(s):** REAL
- **Default:** 1.0e-10

**Description**

The `function_precision` control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.

**absolute_conv_tol**

- Keywords Area
- method
- nl2sol
- absolute_conv_tol

Absolute convergence tolerance

**Specification**

Alias: none

- **Argument(s):** REAL
- **Default:** -1. (use NL2SOL internal default)

**Description**

`absolute_conv_tol` (NL2SOL's `afctol`) is the absolute function convergence tolerance (stop when half the sum of squares is less than `absolute_conv_tol`, which is mainly of interest on zero-residual test problems).

The internal default is a function of machine epsilon (as limited by `function_precision`). The default is selected with a value of -1.
6.2. METHOD

**x_conv_tol**

- Keywords Area
  - method
  - nl2sol
  - x_conv_tol

  X-convergence tolerance

**Specification**

**Alias**: none

**Argument(s)**: REAL

**Default**: -1. (use NL2SOL internal default)

**Description**

*x_conv_tol* maps to the internal NL2SOL control *xctol*. It is the X-convergence tolerance (scaled relative accuracy of the solution variables).

The internal default is a function of machine epsilon (as limited by *function_precision*). The default is selected with a value of -1.

**singular_conv_tol**

- Keywords Area
  - method
  - nl2sol
  - singular_conv_tol

  Singular convergence tolerance

**Specification**

**Alias**: none

**Argument(s)**: REAL

**Default**: -1. (use NL2SOL internal default)

**Description**

*singular_conv_tol* (NL2SOL’s *sctol*) is the singular convergence tolerance, which works in conjunction with *singular_radius* to test for underdetermined least-squares problems (stop when the relative reduction yet possible in the sum of squares appears less then *singular_conv_tol* for steps of scaled length at most *singular_radius*).

The internal default is a function of machine epsilon (as limited by *function_precision*). The default is selected with a value of -1.
**singular_radius**

- Keywords Area
- method
- nl2sol
- singular_radius

Singular radius

### Specification

**Alias:** none  
**Argument(s):** REAL  
**Default:** -1. (use NL2SOL internal default of 1)

### Description

`singular_radius` works in conjunction with `singular_conv_tol` to test for underdetermined least-squares problems (stop when the relative reduction yet possible in the sum of squares appears less than `singular_conv_tol` for steps of scaled length at most `singular_radius`).

The internal default results in the internal use of steps of length 1. The default is selected with a value of -1.

**false_conv_tol**

- Keywords Area
- method
- nl2sol
- false_conv_tol

False convergence tolerance

### Specification

**Alias:** none  
**Argument(s):** REAL  
**Default:** -1. (use NL2SOL internal default)

### Description

`false_conv_tol` (NL2SOL’s `xftol`) is the false-convergence tolerance (stop with a suspicion of discontinuity when a more favorable stopping test is not satisfied and a step of scaled length at most `false_conv_tol` is not accepted).

The internal default is a function of machine epsilon (as limited by `function_precision`). The default is selected with a value of -1.
6.2. METHOD

**initial trust radius**
- **Keywords Area**
- method
- nl2sol
- initial trust radius

Initial trust region radius

**Specification**

**Alias:** none
**Argument(s):** REAL
**Default:** -1. (use NL2SOL internal default of 1)

**Description**

*initial trust radius* specification (NL2SOL’s lmax0) specifies the initial trust region radius for the algorithm.

The internal default results in the internal use of steps of length 1. The default is selected with a value of -1.

**covariance**
- **Keywords Area**
- method
- nl2sol
- covariance

Determine how the final covariance matrix is computed

**Specification**

**Alias:** none
**Argument(s):** INTEGER
**Default:** 0 (no covariance)

**Description**

covariance (NL2SOL’s covreq) specifies whether and how NL2SOL computes a final covariance matrix.

The desired covariance approximation:

- 0 = default = none
- 1 or -1 ==> $\sigma^2 H^{-1} J^T J H^{-1}$
- 2 or -2 ==> $\sigma^2 H^{-1}$
- 3 or -3 ==> $\sigma^2 (J^T J)^{-1}$
• Negative values $\Rightarrow$ estimate the final Hessian H by finite differences of function values only (using \texttt{fd-hessian_step_size})

• Positive values $\Rightarrow$ differences of gradients (using \texttt{fd_hessian_step_size})

\texttt{regression_diagnostics}

• Keywords Area

• method

• nl2sol

• \texttt{regression_diagnostics}

Turn on regression diagnostics

\textbf{Specification}

\textbf{Alias:} none

\textbf{Argument(s):} none

\textbf{Default:} no regression diagnostics

\textbf{Description}

When \texttt{regression_diagnostics} (NL2SOL's \texttt{rdreq}) is specified and a positive-definite final Hessian approximation H is computed, NL2SOL computes and prints a regression diagnostic vector RD such that if omitting the i-th observation would cause alpha times the change in the solution that omitting the j-th observation would cause, then $RD[i] = |alpha| \cdot RD[j]$. The finite-difference step-size tolerance affecting H is \texttt{fd_step_size} (NL2SOL's \texttt{delta0} and \texttt{dltfdc}).

\texttt{convergence_tolerance}

• Keywords Area

• method

• nl2sol

• \texttt{convergence_tolerance}

Stopping criterion based on convergence of the objective function or statistics

\textbf{Topics}

This keyword is related to the topics:

• \texttt{method_independent_controls}

\textbf{Specification}

\textbf{Alias:} none

\textbf{Argument(s):} REAL

\textbf{Default:} 1.e-4
6.2. METHOD

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_iterations

- Keywords Area
- method
- nl2sol
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method-independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)
CHAPTER 6. KEYWORDS AREA

Description

The maximum number of iterations (default: 100). See also max.function_evaluations.

speculative

- Keywords Area
- method
- nl2sol
- speculative

Compute speculative gradients

Topics

This keyword is related to the topics:

- method.independent_controls

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, speculative gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [14] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to responses gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.
The speculative specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The speculative specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for vendor numerical gradients).

max_function_evaluations

- Keywords Area
- method
- nl2sol
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:
- method-independent-controls

Specification

Alias: none

Argument(s): INTEGER
Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling

- Keywords Area
- method
- nl2sol
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:
- method-independent-controls
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and _scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling
   - First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
     - two-sided bounds scaled into the interval [0,1];
     - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
     - no bounds or targets: no automatic scaling possible, therefore no scaling for this component
   - Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. log - logarithmic scaling
   - First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.
   - Logarithmic scaling is not available for linear constraints.
   - When continuous design variables are log scaled, linear constraints are not allowed.
6.2. METHOD

Scales
The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.
Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{-10}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples
The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

model_pointer
- Keywords Area
- method
- nl2sol
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)
Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
```
response_functions = 3
no_gradients
no_hessians

6.2.41 nonlinear_cg

- Keywords Area
- method
- nonlinear_cg

(Experimental) nonlinear conjugate gradient optimization

Topics

This keyword is related to the topics:
- local_optimization_methods

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>misc_options</td>
<td>Options for nonlinear CG optimizer</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_-tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| Optional | model_pointer | Identifier for model block to be used by a method |

Description
This method is an incomplete experimental implementation of nonlinear conjugate gradient optimization, a local, gradient-based solver.

**misc_options**
- Keywords Area
- method
- nonlinear_cg
- misc_options

Options for nonlinear CG optimizer

Specification
**Alias:** none
**Argument(s):** STRINGLIST
**Default:** no misc options

Description
List of miscellaneous string options to pass to the experimental nonlinear CG solver (see NonlinearCGOptimizer-.cpp in the Dakota source code for available controls). Includes controls for step sizes, linesearch control, convergence, etc.

**convergence_tolerance**
- Keywords Area
- method
- nonlinear_cg
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:
- method_independent_controls
6.2. METHOD 887

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration. Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_iterations

- Keywords Area
- method
- nonlinear_cg
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

scaling

- Keywords Area
- method
- nonlinear_cg
- scaling

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and _scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values
6.2. METHOD

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. **none** (default) - no scaling, value of *scales* keyword is ignored
2. **value** - multiplicative scaling
3. **auto** - automatic scaling
   
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   
   - two-sided bounds scaled into the interval [0,1];
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. **log** - logarithmic scaling
   
   First, any characteristic values from the optional *scales* specification are applied. Then logarithm base 10 scaling is applied.

   Logarithmic scaling is not available for linear constraints.

   When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The *scales* keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type*, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than \(1.0 \times 10^{10} \times \text{DBL_MIN}\). User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```
**model_pointer**

- **Keywords Area**
- **method**
- **nonlinear_cg**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

*Alias:* none

*Argument(s):* STRING

*Default:* method use of last model parsed (or use of default model if none parsed)

**Description**

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
```
6.2. METHOD

surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.42 ncsu_direct

• Keywords Area
• method
• ncsu_direct

DIviding RECTangles method

Topics
This keyword is related to the topics:

• global_optimization_methods

Specification

Alias: none

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional | Group | | Description |


CHAPTER 6. KEYWORDS AREA

| Optional | solution_target | Specifies a globally optimal value toward which the optimizer should track |
| Optional | min_boxsize_limit | Stopping Criterion based on shortest edge of hyperrectangle |
| Optional | volume_boxsize_limit | Stopping criterion based on volume of search space |
| Optional | convergence_tolerance | Stopping criterion based on convergence of the objective function or statistics |
| Optional | max_iterations | Stopping criterion based on number of iterations |
| Optional | max_function_evaluations | Stopping criteria based on number of function evaluations |
| Optional | scaling | Turn on scaling for variables, responses, and constraints |
| Optional | model_pointer | Identifier for model block to be used by a method |

Description

North Carolina State University (NCSU) has an implementation of the DIRECT algorithm (DIviding RECTangles algorithm that is outlined in the SCOLIB method section above). This version is documented in [28] We have found that the NCSU DIRECT implementation works better and is more robust for some problems than coliny-direct. Currently, we maintain both versions of DIRECT in Dakota; in the future, we may deprecate one.

The NCSU DIRECT method is selected with ncsu_direct. We have tried to maintain consistency between the keywords in SCOLIB and NCSU implementation of DIRECT, but the algorithms have different parameters, so the keywords sometimes have slightly different meaning.

Stopping Criteria

The algorithm stops based on:

1. **max_iterations** - number of iterations
2. **max_function_evaluations** - number of function evaluations
6.2. METHOD

3. solution_target and convergence_tolerance

4. min_boxsize_limit

5. volume_boxsize_limit

This method will always strictly respect the number of iterations, but may slightly exceed the number of function evaluations, as it will always explore all sub-rectangles at the current level.

See Also

These keywords may also be of interest:

- coliny_direct

solution_target

- Keywords Area
- method
- ncsu_direct
- solution_target

Specifies a globally optimal value toward which the optimizer should track.

Specification

Alias: solution_accuracy
Argument(s): REAL
Default: 0

Description

The solution target specifies a goal toward which the optimizer should track. This is used for test problems, when the true global minimum is known (call it solution_target := fglobal). Then, the optimization terminates when 100(f_min-fglobal)/max(1,abs(fglobal)) < convergence_tolerance. The default for fglobal is -1.0e100 and the default for convergence tolerance is described at convergence_tolerance.

min_boxsize_limit

- Keywords Area
- method
- ncsu_direct
- min_boxsize_limit

Stopping Criterion based on shortest edge of hyperrectangle
CHAPTER 6. KEYWORDS AREA

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 1.0e-4

**Description**

`min_boxsize_limit` is a setting that terminates the optimization when the measure of a hyperrectangle $S$ with $f(c(S)) = f_{\text{min}}$ is less than `min_boxsize_limit`.

Each subregion considered by DIRECT has a size, which corresponds to the longest diagonal of the subregion. `min_boxsize_limit` specification terminates DIRECT if the size of the smallest subregion falls below this threshold.

In practice, this specification is likely to be more effective at limiting DIRECT’s search.

**volume_boxsize_limit**

- Keywords Area
- method
- ncsu_direct
- volume_boxsize_limit

Stopping criterion based on volume of search space

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** 1.0e-6

**Description**

`volume_boxsize_limit` is a setting that terminates the optimization when the volume of a hyperrectangle $S$ with $f(c(S)) = f_{\text{min}}$ is less than `volume_boxsize_limit` percent of the original hyperrectangle. Basically, `volume_boxsize_limit` stops the optimization when the volume of the particular rectangle which has $f_{\text{min}}$ is less than a certain percentage of the whole volume.

**convergence_tolerance**

- Keywords Area
- method
- ncsu_direct
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics
6.2. METHOD

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: 1.e-4

Description
The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration. Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library
This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_iterations
- Keywords Area
- method
- ncsu_direct
- max_iterations

Stopping criterion based on number of iterations
Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global, {reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

max_function_evaluations
- Keywords Area
- method
- ncsu_direct
- max_function_evaluations
  Stopping criteria based on number of function evaluations

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max_iterations.

scaling
- Keywords Area
- method
- ncsu_direct
- scaling
  Turn on scaling for variables, responses, and constraints
6.2. METHOD

Topics

This keyword is related to the topics:

- method
- independent
- controls

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all scale types and *scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].
4. log - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales
The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above. Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{-12}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples
The two examples below are equivalent:

```plaintext
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

model_pointer

- Keywords Area
- method
- ncsu_direct
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer
6.2. METHOD

Specification

Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'Surr'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                        0.1 0.2 0.6
                        0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'Surr'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'
```
interface
   id_interface = 'I1'
   system async evaluation_concurrency = 5
   analysis_driver = 'text_book'

responses
   response_functions = 3
   no_gradients
   no_hessians

6.2.43 genie_opt_darts

   - Keywords Area
   - method
   - genie_opt_darts

Voronoi-based high-dimensional global Lipschitzian optimization

**Specification**

**Alias:** none

**Argument(s):** none

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword Description |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_function_-_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

OPT-Darts method is a fast alternative to DIRECT for global Lipschitzian optimization purposes. Instead of hyperrectangular, OPT-Darts decomposes a high-dimensional domain into Voronoi cells, and places samples via stochastic blue noise instead of deterministic cell division.

To refine a cell, OPT-Darts first adds a new sample within it via spoke-dart sampling, then set the conflict radius to the cells inscribed hypersphere radius, to avoid adding a sample point that is too close to a prior sample, then divide that cell (and update its neighboring cells) via the approximate Delaunay graph, and use the computed witnesses to decide the next refinement candidate. These two steps replace the corresponding deterministic center-sample and rectangular cell division in DIRECT, respectively.
OPT-Darts is the first exact stochastic Lipschitzian optimization technique that combines the benefits of guaranteed convergence in [Jones et al. 1993] and high dimensional efficiency in [Spall 2005]. Computing blue noise and Voronoi regions has been intractable in high dimensions, and are being done within OPT-Darts using Spoke-Darts.

**seed**
- **Keywords Area**
- **method**
- **genie_opt_darts**
- **seed**
  
  Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)

**Description**

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**max_function_evaluations**

- **Keywords Area**
- **method**
- **genie_opt_darts**
- **max_function_evaluations**
  
  Stopping criteria based on number of function evaluations
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:
- method_independent.controls

Specification
Alias: none
Argument(s): INTEGER
Default: 1000

Description
The maximum number of function evaluations (default: 1000). See also max.iterations.

scaling
- Keywords Area
- method
- genie_opt.darts
- scaling

Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by by providing the scaling keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the scaling keyword is omitted, all _scale_types and _scales specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.
6.2. METHOD

- a *scales keyword, which gives characteristic values
- a *scale_type keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**

There are four scale types:

1. none (default) - no scaling, value of *scales keyword is ignored
2. value - multiplicative scaling
3. auto - automatic scaling
   
   First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:
   
   - two-sided bounds scaled into the interval [0,1];
   - one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
   - no bounds or targets: no automatic scaling possible, therefore no scaling for this component

   Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. log - logarithmic scaling
   
   First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.
   
   Logarithmic scaling is not available for linear constraints.
   
   When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10\cdot\text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.
Examples
The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

**model_pointer**

- Keywords Area
- method
- genie_opt_darts
- model_pointer

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- block_pointer

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The **model_pointer** is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.

See block_pointer for details about pointers.
6.2. METHOD

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'Surr'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
  0.1 0.2 0.6
  0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'Surr'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.0
  upper_bounds = 1. 1.0
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.44 genie_direct

- Keywords Area
- method
- genie_direct

Classical high-dimensional global Lipschitzian optimization Classical high-dimensional global Lipschitzian optimization
Specification

Alias: none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_function_evaluations</td>
<td>Stopping criteria based on number of function evaluations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>scaling</td>
<td>Turn on scaling for variables, responses, and constraints</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

Description

DIRECT (DIviding RECTangles) partitions the domain into hyperrectangles and uses an iterative Lipschitzian optimization approach to search for a global optimal point.

DIRECT begins by scaling the domain into the unit hypercube by adopting a center-sampling strategy. The objective function is evaluated at the midpoint of the domain, where a lower bound is constructed. In one-dimension, the domain is tri-sected and two new center points are sampled. At each iteration (dividing and sampling), DIRECT identifies intervals that contain the best minimal value of the objective function found up to that point. This strategy of selecting and dividing gives DIRECT its performance and convergence properties compared to other deterministic methods.

The classical DIRECT method [Shubert 1972] has two limitations: poor scaling to high dimensions; and relying on a global K; whose exact value is often unknown. The enhanced DIRECT algorithm [Jones et al. 1993] generalizes [Shubert 1972] to higher dimensions and does not require knowledge of the Lipschitz constant.

seed

- Keywords Area
- method
- genie_direct
- seed

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)
6.2. **METHOD**

**Description**

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**max_function_evaluations**

- **Keywords Area**
- **method**
- **genie_direct**
- **max_function_evaluations**

Stopping criteria based on number of function evaluations

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 1000

**Description**

The maximum number of function evaluations (default: 1000). See also `max_iterations`. 
Turn on scaling for variables, responses, and constraints

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none
Argument(s): none
Default: no scaling

Description
Some of the optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated by providing the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords which are found in the method, variables, and responses blocks. When the `scaling` keyword is omitted, all `*_scales` and `*_type` specifications are ignored in the method, variables, and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities to be scaled. These quantities can be constraint equations, variables, or responses.

- a `*_scales` keyword, which gives characteristic values
- a `*_type` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can either be zero, one, or equal to the number of quantities to be scaled. If one argument is given, it will apply to all quantities in the set. See the examples below.

**Scale Types**
There are four scale types:

1. `none` (default) - no scaling, value of `*_scales` keyword is ignored
2. `value` - multiplicative scaling
6.2. METHOD

3. **auto** - automatic scaling

First the quantity is scaled by the characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Futher, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

4. **log** - logarithmic scaling

First, any characteristic values from the optional *scales specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

**Scales**

The *scales keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding *scale_type, as described above.

Depending on the scale type, the characteristic values may be required or optional.

- none, auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

**Examples**

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100

responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```
model_pointer

- Keywords Area
- method
- genie_direct
- model_pointer

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- block_pointer

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
        0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```
6.2. METHOD

surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.45 efficient_global

- Keywords Area
- method
- efficient_global

Global Surrogate Based Optimization, a.k.a. EGO

Topics
This keyword is related to the topics:

- global_optimization_methods
- surrogate_based_optimization_methods

Specification

Alias: none

Argument(s): none

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Description**

The Efficient Global Optimization (EGO) method was first developed by Jones, Schonlau, and Welch[54]. In EGO, a stochastic response surface approximation for the objective function is developed based on some sample points from the "true" simulation.

Note that several major differences exist between our implementation and that of[54]. First, rather than using a branch and bound method to find the point which maximizes the EIF, we use the DIRECT global optimization method.

Second, we support both global optimization and global nonlinear least squares as well as general nonlinear constraints through abstraction and subproblem recasting.

The efficient global method is in prototype form. Currently, we do not expose any specification controls for the underlying Gaussian process model used or for the optimization of the expected improvement function (which is currently performed by the NCSU DIRECT algorithm using its internal defaults).

By default, EGO uses the Surfpack GP (Kriging) model, but the Dakota implementation may be selected instead. If `use_derivatives` is specified the GP model will be built using available derivative data (Surfpack GP only).

**Theory**

The particular response surface used is a Gaussian process (GP). The GP allows one to calculate the prediction at a new input location as well as the uncertainty associated with that prediction. The key idea in EGO is to maximize the Expected Improvement Function (EIF). The EIF is used to select the location at which a new training point

---

| Optional | gaussian_process | Gaussian Process surrogate model |
| Optional | use_derivatives | Use derivative data to construct surrogate models |
| Optional | import_build_points_file | File containing points you wish to use to build a surrogate |
| Optional | export_approx_points_file | Output file for evaluations of a surrogate model |
| Optional | seed | Seed of the random number generator |
| Optional | max_iterations | Stopping criterion based on number of iterations |
| Optional | model_pointer | Identifier for model block to be used by a method |
should be added to the Gaussian process model by maximizing the amount of improvement in the objective function that can be expected by adding that point. A point could be expected to produce an improvement in the objective function if its predicted value is better than the current best solution, or if the uncertainty in its prediction is such that the probability of it producing a better solution is high. Because the uncertainty is higher in regions of the design space with few observations, this provides a balance between exploiting areas of the design space that predict good solutions, and exploring areas where more information is needed. EGO trades off this "exploitation vs. exploration." The general procedure for these EGO-type methods is:

- Build an initial Gaussian process model of the objective function
- Find the point that maximizes the EIF. If the EIF value at this point is sufficiently small, stop.
- Evaluate the objective function at the point where the EIF is maximized. Update the Gaussian process model using this new point.
- Return to the previous step.

See Also
These keywords may also be of interest:
- surrogate_based_local
- surrogate_based_global

gaussian_process
- Keywords Area
- method
- efficient_global
- gaussian_process

Gaussian Process surrogate model

Specification
Alias: kriging
Argument(s): none
Default: Surfpack Gaussian process

| Required/-| Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional | Group 1 | surfpack | Use the Surfpack version of Gaussian Process surrogates |
Select the built-in Gaussian Process surrogate

**Description**

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the `surfpack` keyword. An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

```
surfpack
  • Keywords Area
  • method
  • efficient_global
  • gaussian_process
  • surfpack

Use the Surfpack version of Gaussian Process surrogates
```

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack. Several user options are available:

1. **Optimization methods:**
   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default, the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See `optimization_method`.

   The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the “truth” model, and is relatively inexpensive to compute.

2. **Trend Function:**
   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See `trend`.

3. **Correlation Lengths:**
   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See `correlation_lengths`.
4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The surfpack model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here “inexpensive” means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

dakota

- Keywords Area
- method
- efficient_global
- gaussian_process
- dakota

Select the built in Gaussian Process surrogate

Specification

Alias: none
Argument(s): none
Description

A second version of GP surrogates was available in prior versions of Dakota. For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the guassian_process dakota model can be found in[58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a “nugget,” but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

use_derivatives

- Keywords Area
- method
- efficient_global
- use_derivatives

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The use_derivatives flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it’s use with Surfpack Gaussian process is not recommended.

import_build_points_file

- Keywords Area
- method
- efficient_global
6.2. METHOD

- import_build_points_file

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file

**Argument(s):** STRING

**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td></td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The import_build_points_file allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the import_build_points_file must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**Examples**

```
method
  polynomial_chaos
    expansion_order = 4
  import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```
annotated

- Keywords Area
- method
- efficient_global
- import_build_points_file
- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
6.2. METHOD

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

custom_annotated

- Keywords Area
- method
- efficient_global
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id  x1    x2    obj_fn  nln_ineq_con_1  nln_ineq_con_2
1  0.9    1.1    0.0002  0.26       0.76
2  0.90009 1.1  0.0001996404857 0.2601620081 0.759955
3  0.89991 1.1  0.0002003604863 0.2598380081 0.760045
...```

header

- Keywords Area
- method
- efficient_global
6.2. METHOD

- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none
- **Argument(s):** none
- **Default:** no header

**Description**

See description of parent custom_annotated

- **eval_id**
  - Keywords Area
  - method
  - efficient_global
  - import_build_points_file
  - custom_annotated
  - eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none
- **Argument(s):** none
- **Default:** no eval_id column

**Description**

See description of parent custom_annotated

- **interface_id**
  - Keywords Area
  - method
  - efficient_global
  - import_build_points_file
  - custom_annotated
  - interface_id

Enable interface ID column in custom-annotated tabular file
**Specification**

Alias: none  
Argument(s): none  
Default: no interface_id column

**Description**

See description of parent `custom.annotated`

**freeform**

- Keywords Area  
- method  
- efficient_global  
- import_build_points_file  
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none  
Argument(s): none  
Default: annotated format

**Description**

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
In **freeform**, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the **freeform** option.

**Examples**

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```python
environment
tabular_data = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

**active_only**

- Keywords Area
- method
- efficient_global
- import_build.points_file
- active_only

Import only active variables from tabular data file

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

*Alias:* none

*Argument(s):* none

**Description**

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword **active_only** indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.
**export_approx_points_file**

- **Keywords Area**
- **method**
- **efficient_global**
- **export_approx_points_file**

Output file for evaluations of a surrogate model

### Specification

**Alias:** export_points_file  
**Argument(s):** STRING  
**Default:** no point export to a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

### Description

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.

**Usage Tips**

Dakota exports tabular data in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**annotated**

- **Keywords Area**
- **method**
- **efficient_global**
- **export_approx_points_file**
6.2. METHOD

- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated_header eval_id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```
custom.annotated

- Keywords Area
- method
- efficient_global
- export_approx.points_file
- custom.annotated

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- file.formats

**Specification**

*Alias:* none  
*Argument(s):* none  
*Default:* annotated format

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom.annotated, followed by options, in the relevant export/import context.

**Usage Tips**
• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
    tabular_data
        tabular_data_file = ‘dakota_summary.dat’
        custom_annotated header eval_id

Resulting tabular file:

%eval_id x1  x2  obj_fn nln_ineq_con_1 nln_ineq_con_2
1  0.9  1.1  0.0002  0.26  0.76
2  0.90009 1.1  0.0001996404857  0.2601620081  0.759955
3  0.89991 1.1  0.0002003604863  0.2598380081  0.760045
...
```

header

• Keywords Area

• method

• efficient_global

• export_approx_points_file

• custom_annotated

• header

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent custom_annotated
CHAPTER 6. KEYWORDS AREA

- eval
  - Keywords Area
  - method
  - efficient_global
  - export_approx_points_file
  - custom.annotated
  - eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no eval_id column

**Description**

See description of parent `custom.annotated`

- interface_id
  - Keywords Area
  - method
  - efficient_global
  - export_approx_points_file
  - custom.annotated
  - interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no interface_id column

**Description**

See description of parent `custom.annotated`
6.2. METHOD

freeform

- Keywords Area
- method
- efficient_global
- export_approx_points_file
- freeform

Selects freeform file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
  Argument(s): none
  Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
freeform
Resulting tabular file:

0.9  1.1  0.0002  0.26  0.76
0.90009  1.1  0.0001996404857  0.2601620081  0.759955
0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

seed

- Keywords Area
- method
- efficient_global
- seed

Seed of the random number generator

Specification

Alias: none
Argument(s): INTEGER
Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```
6.2. METHOD

max_iterations

- Keywords Area
- method
- efficient_global
- max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:
- method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

model_pointer

- Keywords Area
- method
- efficient_global
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:
- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)
CHAPTER 6. KEYWORDS AREA

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
    sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
```
response_functions = 3
no_gradients
no_hessians

6.2.46 polynomial_choas

- Keywords Area
- method
- polynomial_choas

Uncertainty quantification using polynomial chaos expansions

Specification

**Alias:** nond_polynomial_choas

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>p_refinement</td>
<td>Automatic polynomial order refinement</td>
</tr>
<tr>
<td>Optional/Choose One</td>
<td>Basis polynomial family (Group 1)</td>
<td>askey</td>
<td>Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wiener</td>
<td>Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.</td>
</tr>
</tbody>
</table>
## Chapter 6. Keywords Area

### Required (Choose One)

<table>
<thead>
<tr>
<th>Coefficient estimation approach (Group 2)</th>
<th>quadrature_order</th>
<th>Cubature using tensor-products of Gaussian quadrature rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>sparse_grid_level</td>
<td></td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
<tr>
<td>cubature_integrand</td>
<td></td>
<td>Cubature using Stroud rules and their extensions</td>
</tr>
<tr>
<td>expansion_order</td>
<td></td>
<td>The (initial) order of a polynomial expansion</td>
</tr>
<tr>
<td>orthogonal_least_interpolation</td>
<td></td>
<td>Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.</td>
</tr>
<tr>
<td>import_expansion_file</td>
<td></td>
<td>Build a Polynomial Chaos Expansion (PCE) by import coefficients and a multi-index from a file</td>
</tr>
</tbody>
</table>

### Optional

<table>
<thead>
<tr>
<th>variance_based_decomp</th>
<th>Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>Covariance type (Group 3)</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td></td>
<td>full_covariance</td>
</tr>
<tr>
<td>Optional</td>
<td>normalized</td>
</tr>
<tr>
<td>Optional</td>
<td>sample_type</td>
</tr>
<tr>
<td>Optional</td>
<td>probability_refinement</td>
</tr>
<tr>
<td>Optional</td>
<td>import_approx_points_file</td>
</tr>
<tr>
<td>Optional</td>
<td>export_approx_points_file</td>
</tr>
<tr>
<td>Optional</td>
<td>export_expansion_file</td>
</tr>
</tbody>
</table>
### Optional Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed_seed</td>
<td>Reuses the same seed value for multiple random sampling sets</td>
</tr>
<tr>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>reliability_levels</td>
<td>Specify reliability levels at which the response values will be estimated. Values at which to estimate desired statistics for each response.</td>
</tr>
<tr>
<td>response_levels</td>
<td>Selection of cumulative or complementary cumulative functions.</td>
</tr>
<tr>
<td>distribution</td>
<td>Specify probability levels at which to estimate the corresponding response value.</td>
</tr>
<tr>
<td>probability_levels</td>
<td>Specify generalized reliability levels at which to estimate the corresponding response value.</td>
</tr>
</tbody>
</table>
6.2. METHOD

Optional

rng
Selection of a random number generator

Optional

samples
Number of samples for sampling-based methods

Optional

seed
Seed of the random number generator

Optional

model_pointer
Identifier for model block to be used by a method

Description

The polynomial chaos expansion (PCE) is a general framework for the approximate representation of random response functions in terms of finite-dimensional series expansions in standardized random variables

$$R = \sum_{i=0}^{P} \alpha_i \Psi_i(\xi)$$

where $\alpha_i$ is a deterministic coefficient, $\Psi_i$ is a multidimensional orthogonal polynomial and $\xi$ is a vector of standardized random variables. An important distinguishing feature of the methodology is that the functional relationship between random inputs and outputs is captured, not merely the output statistics as in the case of many nondeterministic methodologies.

Basis polynomial family (Group 1)

Group 1 keywords are used to select the type of basis, $\Psi_i$, of the expansion. Three approaches may be employed:

- Wiener: employs standard normal random variables in a transformed probability space, corresponding to Hermite orthogonal basis polynomials (see wiener).
- Askey: employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space, corresponding to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal basis polynomials, respectively (see askey).
- Extended (default if no option is selected): The Extended option avoids the use of any nonlinear variable transformations by augmenting the Askey approach with numerically-generated orthogonal polynomials for non-Askey probability density functions. Extended polynomial selections replace each of the sub-optimal Askey basis selections for bounded normal, lognormal, bounded lognormal, loguniform, triangular, gumbel, frechet, weibull, and bin-based histogram.

For supporting correlated random variables, certain fallbacks must be implemented.

- The Extended option is the default and supports only Gaussian correlations.
- If needed to support prescribed correlations (not under user control), the Extended and Askey options will fall back to the Wiener option on a per variable basis. If the prescribed correlations are also unsupported by Wiener expansions, then Dakota will exit with an error.
Coefﬁcient estimation approach (Group 2)
To obtain the coefﬁcients \( \alpha_i \) of the expansion, seven options are provided:

1. multidimensional integration by a tensor-product of Gaussian quadrature rules (speciﬁed with quadrature_order, and, optionally, dimension_preference).

2. multidimensional integration by the Smolyak sparse grid method (speciﬁed with sparse_grid_level and, optionally, dimension_preference).

3. multidimensional integration by Stroud cubature rules and extensions as speciﬁed with cubature_integrand.

4. multidimensional integration by Latin hypercube sampling (speciﬁed with expansion_order and expansion_samples).

5. linear regression (speciﬁed with expansion_order and either collocation_points or collocation_ratio), using either over-determined (least squares) or under-determined (compressed sensing) approaches.

6. orthogonal least interpolation (speciﬁed with orthogonal_least_interpolation and collocation_points).

7. coefﬁcient import from a file (speciﬁed with import_expansion_file). The expansion can be comprised of a general set of expansion terms, as indicated by the multi-index annotation within the file.

It is important to note that, while quadrature_order, sparse_grid_level, and expansion_order are array inputs, only one scalar from these arrays is active at a time for a particular expansion estimation. These scalars can be augmented with a dimension_preference to support anisotropy across the random dimension set. The array inputs are present to support advanced use cases such as multifidelity UQ, where multiple grid resolutions can be employed.

Active Variables
The default behavior is to form expansions over aleatory uncertain continuous variables. To form expansions over a broader set of variables, one needs to specify active followed by state, epistemic, design, or all in the variables speciﬁcation block.

For continuous design, continuous state, and continuous epistemic uncertain variables included in the expansion, Legendre chaos bases are used to model the bounded intervals for these variables. However, these variables are not assumed to have any particular probability distribution, only that they are independent variables. Moreover, when probability integrals are evaluated, only the aleatory random variable domain is integrated, leaving behind a polynomial relationship between the statistics and the remaining design/state/epistemic variables.

Covariance type (Group 3)
These two keywords are used to specify how this method computes, stores, and outputs the covariance of the responses. In particular, the diagonal covariance option is provided for reducing post-processing overhead and output volume in high dimensional applications.

Optional Keywords regarding method outputs
Each of these sampling speciﬁcations refer to sampling on the PCE approximation for the purposes of generating approximate statistics.

- sample_type
- samples
- seed
- fixed_seed
which should be distinguished from simulation sampling for generating the PCE coefficients as described in options 4, 5, and 6 above (although these options will share the sample_type, seed, and rng settings, if provided). When using the probability_refinement control, the number of refinement samples is not under the user’s control (these evaluations are approximation-based, so management of this expense is less critical). This option allows for refinement of probability and generalized reliability results using importance sampling.

**Multi-fidelity UQ**
The advanced use case of multifidelity UQ automatically becomes active if the model selected for iteration by the method specification is a multifidelity surrogate model (see hierarchical). In this case, an expansion will first be formed for the model discrepancy (the difference between response results if additive correction or the ratio of results if multiplicative correction), using the first quadrature_order, sparse_grid_level, or expansion_order value along with any specified refinement strategy. Second, an expansion will be formed for the low fidelity surrogate model, using the second quadrature_order, sparse_grid_level, or expansion_order value (if present; the first is reused if not present) along with any specified refinement strategy. Then the two expansions are combined (added or multiplied) into an expansion that approximates the high fidelity model, from which the final set of statistics are generated. For polynomial chaos expansions, this high fidelity expansion can differ significantly in form from the low fidelity and discrepancy expansions, particularly in the multiplicative case where it is expanded to include all of the basis products.

**Usage Tips**
If \( n \) is small (e.g., two or three), then tensor-product Gaussian quadrature is quite effective and can be the preferred choice. For moderate to large \( n \) (e.g., five or more), tensor-product quadrature quickly becomes too expensive and the sparse grid and regression approaches are preferred. Random sampling for coefficient estimation is generally not recommended due to its slow convergence rate. For incremental studies, approaches 4 and 5 support reuse of previous samples through the incremental_lhs and reuse_points specifications, respectively.

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of \( m = 2l + 1 \) are used to enforce odd quadrature orders, where \( l \) is the grid level and \( m \) is the number of points in the rule. The precision of this Gauss rule is then \( i = 2m - 1 = 4l + 1 \). For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level \( l \) for a sparse grid (\( i = 4l + 1 \)) or an order \( m \) for a tensor grid (\( i = 2m - 1 \)). This behavior is known as “restricted growth” or “delayed sequences.” To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive_p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

**Additional Resources**
Dakota provides access to PCE methods through the NonDPolynomialChaos class. Refer to the Uncertainty
Quantification Capabilities chapter of the Users Manual[4] and the Stochastic Expansion Methods chapter of the

**Examples**

```plaintext
method, polynomial_chaos
  sparse_grid_level = 2
  samples = 10000 seed = 12347 rng rnum2
  response_levels = .1 1. 50. 100. 500. 1000.
  variance_based_decomp
```

**See Also**

These keywords may also be of interest:

- adaptive_sampling
- gpais
- local_reliability
- global_reliability
- sampling
- importance_sampling
- stoch_collocation

**p_refinement**

- Keywords Area
- method
- polynomial_chaos
- p_refinement

Automatic polynomial order refinement

**Specification**

Alias: none

Argument(s): none

Default: no refinement

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Required (Choose One)</th>
<th>p-refinement type (Group 1)</th>
<th>uniform</th>
<th>Refine an expansion uniformly in all dimensions.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dimension_adaptive</td>
<td></td>
<td>Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher ‘importance’.</td>
</tr>
</tbody>
</table>

**Description**

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol’ indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User’s Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

**uniform**

- Keywords Area
Refine an expansion uniformly in all dimensions.

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The quadrature order or sparse grid level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion order is ramped by one on each iteration while the oversampling ratio (either defined by collocation ratio or inferred from collocation points based on the initial expansion) is held fixed.

**dimension_adaptive**

- **Keywords Area**

- method

- polynomial_chaos

- p_refinement

- uniform

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher ‘importance’.

**Specification**

**Alias:** none

**Argument(s):** none
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group dimension adaptivity estimation approach (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td>sobol</td>
<td>Estimate dimension preference for automated refinement of stochastic expansion using total Sobol’ sensitivity indices.</td>
</tr>
<tr>
<td>decay</td>
<td>Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>generalized</td>
<td>Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher ‘importance’ in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

- **sobol**
  - Keywords Area
  - method
  - polynomial_chaos
  - p_refinement
  - dimension_adaptive
  - sobol

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol’ sensitivity indices.
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): none
  Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method, polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4

decay
```

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none
  Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method, polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```
6.2. METHOD

**generalized**
- Keywords Area
- method
- polynomial_chaos
- p_refinement
- dimension_adaptive
- generalized

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User’s Manual PCE descriptions for additional specifics).

**Examples**

```
method,  
polynomial_chaos
  sparse_grid_level = 3  
  dimension_adaptive p_refinement generalized  
  max_iterations = 20  
  convergence_tol = 1.e-4
```

**askey**

- Keywords Area
- method
- polynomial_chaos
- askey

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** extended (Askey + numerically-generated)
**Description**

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

**See Also**

These keywords may also be of interest:

- polynomial.chaos
- wiener

**wiener**

- Keywords Area
- method
- polynomial.chaos
- wiener

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

**Specification**

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

**Description**

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

**See Also**

These keywords may also be of interest:

- polynomial.chaos
- askey
6.2. METHOD

quadrature_order

- Keywords Area
- method
- polynomial_chaos
- quadrature_order

Cubature using tensor-products of Gaussian quadrature rules

Specification

Alias: none

**Argument(s):** INTEGERLIST

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>dimension_preference</td>
<td>A set of weights specifying the relative importance of each uncertain variable (dimension)</td>
</tr>
<tr>
<td>Optional/Choose One</td>
<td>Nesting of quadrature rules (Group 1)</td>
<td>nested</td>
<td>Enforce use of nested quadrature rules if available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non_nested</td>
<td>Enforce use of non-nested quadrature rules</td>
</tr>
</tbody>
</table>

**Description**

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with quadrature_order, and, optionally, dimension_preference). The default rule selection is to employ non_nested Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of p_refinement or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a dimension_preference, where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified quadrature_order and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order $p_i$ of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, $p_i$ is one less than the quadrature order $m_i$ in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, $N$, in a tensor-product expansion involving $n$ uncertain input variables is

$$N = 1 + P = \prod_{i=1}^{n} (p_i + 1)$$
In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the quadrature_order specification supports an array input.

**dimension_preference**
- Keywords Area
- method
- polynomial_chaos
- quadrature_order
- dimension_preference

A set of weights specifying the relative importance of each uncertain variable (dimension)

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** isotropic grids

**Description**

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

**See Also**

These keywords may also be of interest:
- sobol
- decay

**nested**

- Keywords Area
- method
- polynomial_chaos
- quadrature_order
- nested

Enforce use of nested quadrature rules if available

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** quadrature: non_nested unless automated refinement; sparse grids: nested
6.2. METHOD

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

non_nested

- Keywords Area
- method
- polynomial_chaos
- quadrature_order
- non_nested

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

sparse_grid_level

- Keywords Area
- method
- polynomial_chaos
- sparse_grid_level

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation

Specification

Alias: none

Argument(s): INTEGERLIST
### Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for quadrature order, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.
6.2. METHOD

restricted

- Keywords Area
- method
- polynomial_chaos
- sparse_grid_level
- restricted

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of \( m = 2l + 1 \) are used to enforce odd quadrature orders, where \( l \) is the grid level and \( m \) is the number of points in the rule. The precision of this Gauss rule is then \( i = 2m - 1 = 4l + 1 \). For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level \( l \) for a sparse grid (\( i = 4l + 1 \)) or an order \( m \) for a tensor grid (\( i = 2m - 1 \)). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p-refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

unrestricted

- Keywords Area
- method
- polynomial_chaos
- sparse_grid_level
- unrestricted

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where $l$ is the grid level and $m$ is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level $l$ for a sparse grid ($i = 4l + 1$) or an order $m$ for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimensionadaptive p-refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

dimension_preference

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **sparse_grid_level**
- **dimension_preference**

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

**Alias:** none

**Argument(s):** REALLIST

**Default:** isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- **sobol**
- **decay**
6.2. METHOD

nested

- Keywords Area
- method
- polynomial_chaos
- sparse_grid_level
- nested

Enforce use of nested quadrature rules if available

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** quadrature: non_nested unless automated refinement; sparse grids: nested

**Description**

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

non_nested

- Keywords Area
- method
- polynomial_chaos
- sparse_grid_level
- non_nested

Enforce use of non-nested quadrature rules

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule
Cubature using Stroud rules and their extensions

Specification

Alias: none
Argument(s): INTEGER

Description

Multi-dimensional integration by Stroud cubature rules [77] and extensions [90], as specified with \texttt{cubature} - \texttt{integrand}. A total-order expansion is used, where the isotropic order \( p \) of the expansion is half of the integrand order, rounded down. The total number of terms \( N \) for an isotropic total-order expansion of order \( p \) over \( n \) variables is given by

\[
N = 1 + P = 1 + \sum_{s=1}^{p} \frac{1}{s!} \prod_{r=0}^{s-1} (n + r) = \frac{(n + p)!}{n!p!}
\]

Since the maximum integrand order is currently five for normal and uniform and two for all other types, at most second- and first-order expansions, respectively, will be used. As a result, cubature is primarily useful for global sensitivity analysis, where the Sobol’ indices will provide main effects and, at most, two-way interactions. In addition, the random variable set must be independent and identically distributed (\textit{iid}), so the use of \texttt{askey} or \texttt{wiener} transformations may be required to create \textit{iid} variable sets in the transformed space (as well as to allow usage of the higher order cubature rules for normal and uniform). Note that global sensitivity analysis often assumes uniform bounded regions, rather than precise probability distributions, so the \textit{iid} restriction would not be problematic in that case.

expansion \_order

Alias: none
Argument(s): INTEGERLIST
### Required/Optional Description of Group Dakota Keyword Dakota Keyword Description

<table>
<thead>
<tr>
<th>Description</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>dimension_preference</td>
<td>basis_type</td>
<td>A set of weights specifying the relative importance of each uncertain variable (dimension) Specify the type of truncation to be used with a Polynomial Chaos Expansion. Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.</td>
</tr>
<tr>
<td>Optional</td>
<td>collocation_points</td>
<td>collocation_ratio</td>
<td>Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>expansion_samples</td>
<td>import_build_points_file</td>
<td>The Number simulation samples to estimate the PCE coefficients File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>

### Description

When the expansion_order for a a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with expansion_samples). In this case, the expansion order $p$ cannot be inferred from the numerical integration specification and it is necessary to provide an expansion_order to specify $p$ for a total-order expansion.

Linear regression (specified with either collocation_points or collocation_ratio). A total-order expansion is used and must be specified using expansion_order as described in the previous option. To avoid requiring the user to calculate $N$ from $n$ and $p$), the collocation_ratio allows for specification of a
constant factor applied to $N$ (e.g., $\text{collocation\_ratio} = 2.$ produces samples $= 2N$). In addition, the default linear relationship with $N$ can be overridden using a real-valued exponent specified using $\text{ratio\_order}$. In this case, the number of samples becomes $cN^o$ where $c$ is the $\text{collocation\_ratio}$ and $o$ is the $\text{ratio\_order}$. The $\text{use\_derivatives}$ flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{o+1}$). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the $\text{tensor\_grid}$ keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If $\text{collocation\_points}$ or $\text{collocation\_ratio}$ is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the $\text{noise\_tolerance}$, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders (1,...,$\text{expansion\_order}$) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

dimension\_preference

- Keywords Area
- method
- polynomial\_chaos
- expansion\_order
- dimension\_preference

A set of weights specifying the relative importance of each uncertain variable (dimension)

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** isotropic grids

**Description**

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.
See Also

These keywords may also be of interest:

- sobol
- decay

basis_type

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- basis_type

Specify the type of truncation to be used with a Polynomial Chaos Expansion.

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required*(Choose One)*</td>
<td></td>
<td>tensor_product</td>
<td>Use a tensor-product index set to construct a polynomial chaos expansion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>total_order</td>
<td>Use a total-order index set to construct a polynomial chaos expansion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adapted</td>
<td>Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.</td>
</tr>
</tbody>
</table>

Description

Specify the type of truncation to be used with a Polynomial Chaos Expansion.
Use a tensor-product index set to construct a polynomial chaos expansion.

**Specification**

Alias: none

**Argument(s):** none

**Description**

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order $p$ keep all terms with $d$-dimensional multi index $\mathbf{i} = (i_1, \ldots, i_d)$ that satisfies

$$\max (i_1, \ldots, i_d) \leq p$$

Use a total-order index set to construct a polynomial chaos expansion.

**Specification**

Alias: none

**Argument(s):** none
6.2. METHOD

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order $p$ keep all terms with a $d$-dimensional multi index $\mathbf{i} = (i_1, \ldots, i_d)$ that satisfies

$$\sum_{k=1}^{d} i_k \leq p$$

adapted

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- basis_type
- adapted

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>advancements</td>
<td>The maximum number of steps used to expand a basis step.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>soft_convergence_-limit</td>
<td>The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.</td>
</tr>
</tbody>
</table>

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify an initial set of non-zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.
advancements

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- basis_type
- adapted
- advancements

The maximum number of steps used to expand a basis step.

**Specification**

Alias: none  
Argument(s): INTEGER

**Description**

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify an initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

**See Also**

These keywords may also be of interest:

- adapted

**soft_convergence_limit**

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- basis_type
- adapted
- soft_convergence_limit

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.
6.2. METHOD

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

See Also

These keywords may also be of interest:

- adapted

collocation_points

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points

Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.

Specification

Alias: none

Argument(s): INTEGERLIST
### Dakota Keyword Description

**Optional**

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>ratio_order</td>
<td>Specify a non-linear relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>Group 1</td>
<td>least_squares</td>
<td>Compute the coefficients of a polynomial expansion using least squares</td>
</tr>
<tr>
<td></td>
<td></td>
<td>orthogonal_matching_pursuit</td>
<td>Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basis_pursuit</td>
<td>Compute the coefficients of a polynomial expansion by solving the Basis Pursuit $\ell_1$-minimization problem using linear programming.</td>
</tr>
</tbody>
</table>
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>basis_pursuit-denoising</strong></td>
<td>Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising $\ell_1$-minimization problem using second order cone optimization.</td>
</tr>
<tr>
<td><strong>least_angle-regression</strong></td>
<td>Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.</td>
</tr>
<tr>
<td><strong>least_absolute-shrinkage</strong></td>
<td>Compute the coefficients of a polynomial expansion by using the LASSO problem.</td>
</tr>
<tr>
<td><strong>cross_validation</strong></td>
<td>Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.</td>
</tr>
<tr>
<td><strong>use_derivatives</strong></td>
<td>Use derivative data to construct surrogate models.</td>
</tr>
<tr>
<td><strong>tensor_grid</strong></td>
<td>Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| Optional     | reuse_points | This describes the behavior of reuse of points in constructing polynomial chaos expansion models. |

**Description**

Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.

\[ \text{ratio\_order} \]

- **Keywords Area**
- **method**
- **polynomial\_chaos**
- **expansion\_order**
- **collocation\_points**
- **ratio\_order**

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

**Specification**

Alias: none  
**Argument(s):** REAL  
**Default:** 1.

**Description**

When using regression type methods (specified with either collocation\_points or collocation\_ratio), a total-order expansion can be specified using expansion\_order. To avoid requiring the user to calculate \( N \) from \( n \) and \( p \), the collocation\_ratio allows for specification of a constant factor applied to \( N \) (e.g., collocation\_ratio = 2. produces samples = 2\( N \)). In addition, the default linear relationship with \( N \) can be overridden using a real-valued exponent specified using ratio\_order. In this case, the number of samples becomes \( cN^{\omega_o} \) where \( c \) is the collocation\_ratio and \( \omega_o \) is the ratio\_order.

\[ \text{least\_squares} \]

- **Keywords Area**
- **method**
- **polynomial\_chaos**
- **expansion\_order**
6.2. METHOD

- collocation_points
- least_squares

Compute the coefficients of a polynomial expansion using least squares

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** svd

<table>
<thead>
<tr>
<th>Required/-Optional/Choose One</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>svd</td>
<td>Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>equality_-constrained</td>
<td>Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.</td>
</tr>
</tbody>
</table>

**Description**

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares.

svd

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- least_squares
- svd

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
Argument(s): none

Description
Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

equality_constrained

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- least_squares
- equality_constrained

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification
Alias: none
Argument(s): none

Description
Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

orthogonal_matching_pursuit

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- orthogonal_matching_pursuit

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification
Alias: omp
Argument(s): none
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>noise_tolerance</td>
<td></td>
</tr>
</tbody>
</table>

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

noise_tolerance

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **expansion_order**
- **collocation_points**
- **orthogonal_matching_pursuit**
- **noise_tolerance**

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

**Alias:** none

**Argument(s):** REALIST

**Default:** 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

basis_pursuit

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **expansion_order**
- **collocation_points**
• **basis_pursuit**

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit $\ell_1$-minimization problem using linear programming.

**Specification**

**Alias:** bp

**Argument(s):** none

**Description**

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit $\ell_1$-minimization problem using linear programming.

```plaintext
basis_pursuit_denoising
```

• **Keywords Area**

• **method**

• **polynomial_chaos**

• **expansion_order**

• **collocation_points**

• **basis_pursuit_denoising**

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising $\ell_1$-minimization problem using second order cone optimization.

**Specification**

**Alias:** bpdn

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><strong>noise_tolerance</strong></td>
<td>The noise tolerance used when performing cross validation in the presence of noise or truncation errors.</td>
</tr>
</tbody>
</table>

**Description**

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising $\ell_1$-minimization problem using second order cone optimization.
6.2. \textbf{METHOD}

\begin{verbatim}
noise_tolerance
  \textbullet\ Keywords Area
  \textbullet\ method
  \textbullet\ polynomial_chaos
  \textbullet\ expansion_order
  \textbullet\ collocation_points
  \textbullet\ basis_pursuit_denoising
  \textbullet\ noise_tolerance

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

\textbf{Specification}

\textbf{Alias:} none

\textbf{Argument(s):} REALLIST

\textbf{Default:} 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

\textbf{Description}

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

\textbf{least_angle_regression}

\textbullet\ Keywords Area

\textbullet\ method

\textbullet\ polynomial_chaos

\textbullet\ expansion_order

\textbullet\ collocation_points

\textbullet\ least_angle_regression

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

\textbf{Specification}

\textbf{Alias:} lars

\textbf{Argument(s):} none

\begin{tabular}{|c|c|c|c|}
  \hline
  Required/- & Description of & Dakota Keyword & Dakota Keyword Description \\
  Optional & Group & & \\
  \hline
  & & & \\
  \hline
\end{tabular}
### Optional

| noise_tolerance | The noise tolerance used when performing cross validation in the presence of noise or truncation errors. |

### Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

#### noise_tolerance

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **expansion_order**
- **collocation_points**
- **least_angle_regression**
- **noise_tolerance**

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

### Specification

**Alias:** none

**Argument(s):** REALIST

**Default:** 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

### Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

#### least_absolute_shrinkage

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **expansion_order**
- **collocation_points**
- **least_absolute_shrinkage**

Compute the coefficients of a polynomial expansion by using the LASSO problem.
6.2. **METHOD**

### Specification

**Alias:** lasso  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>noise_tolerance</td>
<td>The noise tolerance used when performing cross validation in the presence of noise or truncation errors.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>12_penalty</td>
<td>The $l_2$ penalty used when performing compressed sensing with elastic net.</td>
</tr>
</tbody>
</table>

### Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

- **noise_tolerance**
  - **Keywords Area**
  - method
  - polynomial_chaos
  - expansion_order
  - collocation_points
  - least_absolute_shrinkage
  - noise_tolerance

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

### Specification

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

### Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
**12_penalty**

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- least_absolute_shrinkage
- 12_penalty

The $l_2$ penalty used when performing compressed sensing with elastic net.

**Specification**

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

**Description**

The $l_2$ penalty used when performing compressed sensing with elastic net.

**cross_validation**

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- cross_validation

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

**Specification**

Alias: none

Argument(s): none

**Description**

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.
6.2. METHOD

use_derivatives

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- use_derivatives

Use derivative data to construct surrogate models

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** use function values only

**Description**

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it’s use with Surfpack Gaussian process is not recommended.

**tensor_grid**

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- tensor_grid

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** regression with LHS sample set (point collocation)
Description
The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

reuse_points
- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_points
- reuse_points
This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification
Alias: reuse_samples
  Argument(s): none
  Default: no sample reuse in coefficient estimation

Description
The reuse_points option controls the reuse behavior of points for various types of polynomial chaos expansions, including: collocation_points, collocation_ratio, expansion_samples, or orthogonal_least_interpolation. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify reuse_points so that any points that have been previously generated (for example, from the import_points file) can be reused.

colocation_ratio
- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_ratio
  Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification
Alias: none
  Argument(s): REAL
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratio_order</td>
<td>Specify a non-linear relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.</td>
</tr>
<tr>
<td>least_squares</td>
<td>Compute the coefficients of a polynomial expansion using least squares</td>
</tr>
</tbody>
</table>

**Optional (Choose One) Group 1**

<table>
<thead>
<tr>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orthogonal_matching_pursuit</td>
<td>Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)</td>
</tr>
<tr>
<td>basis_pursuit</td>
<td>Compute the coefficients of a polynomial expansion by solving the Basis Pursuit $\ell_1$-minimization problem using linear programming.</td>
</tr>
</tbody>
</table>
Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising $\ell_1$-minimization problem using second order cone optimization.

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion. Use derivative data to construct surrogate models.

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
### Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate $N$ from $n$ and $p$, the collocation_ratio allows for specification of a constant factor applied to $N$ (e.g., collocation_ratio = 2. produces samples = 2N). In addition, the default linear relationship with $N$ can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes $cN^o$ where $c$ is the collocation_ratio and $o$ is the ratio_order. The use_derivatives flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

**ratio_order**

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_ratio
- ratio_order

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

### Specification

**Alias:** none

**Argument(s):** REAL

**Default:** 1.

### Description

When using regression type methods (specified with either collocation_points or collocation_ratio), a total-order expansion can be specified using expansion_order. To avoid requiring the user to calculate $N$ from $n$ and $p$, the collocation_ratio allows for specification of a constant factor applied to $N$ (e.g., collocation_ratio = 2. produces samples = 2N). In addition, the default linear relationship with $N$ can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes $cN^o$ where $c$ is the collocation_ratio and $o$ is the ratio_order.
least_squares

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_ratio
- least_squares

Compute the coefficients of a polynomial expansion using least squares

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** svd

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword svd</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
</table>
| optional (Choose One) | equality_constrained | Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

**Description**

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares.

svd

- Keywords Area
- method
- polynomial_chaos
- expansion_order
6.2. METHOD

- collocation_ratio
- least_squares
- svd

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none
Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

equality_constrained

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_ratio
- least_squares
- equality_constrained

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none
Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.
orthogonal_matching_pursuit

- Keywords Area
- method
- polynomial_choas
- expansion_order
- collocation_ratio
- orthogonal_matching_pursuit

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

**Specification**

**Alias:** omp

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>noise_tolerance</td>
<td>The noise tolerance used when performing cross validation in the presence of noise or truncation errors.</td>
</tr>
</tbody>
</table>

**Description**

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

**noise_tolerance**

- Keywords Area
- method
- polynomial_choas
- expansion_order
- collocation_ratio
- orthogonal_matching_pursuit
- noise_tolerance

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
6.2. METHOD

Specification

Alias: none

Argument(s): REALLIST
Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

\texttt{basis\_pursuit}
- Keywords Area
- method
- polynomial\_chaos
- expansion\_order
- collocation\_ratio
- basis\_pursuit

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit $\ell_1$-minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit $\ell_1$-minimization problem using linear programming.

\texttt{basis\_pursuit\_denoising}
- Keywords Area
- method
- polynomial\_chaos
- expansion\_order
- collocation\_ratio
- basis\_pursuit\_denoising

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising $\ell_1$-minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none
<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>noise_tolerance</td>
<td>The noise tolerance used when performing cross validation in the presence of noise or truncation errors.</td>
</tr>
</tbody>
</table>

**Description**

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising $\ell_1$-minimization problem using second order cone optimization.

- `noise_tolerance`
  - Keywords Area
  - method
  - polynomial_chaos
  - expansion_order
  - collocation_ratio
  - basis_pursuit_denoising
  - noise_tolerance

  The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

**Specification**

*Alias:* none

*Argument(s):* REALLIST

*Default:* 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

**Description**

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

- `least_angle_regression`
  - Keywords Area
  - method
  - polynomial_chaos
  - expansion_order
  - collocation_ratio
  - least_angle_regression

  Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
6.2. METHOD

Specification
Alias: lars
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>noise_tolerance</td>
<td>The noise tolerance used when performing cross validation in the presence of noise or truncation errors.</td>
</tr>
</tbody>
</table>

Description
Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

noise_tolerance
- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_ratio
- least_angle_regression
- noise_tolerance

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification
Alias: none
Argument(s): REALLIST
Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description
The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

least_absolute_shrinkage
- Keywords Area
- method
- polynomial_chaos
• expansion_order
• collocation_ratio
• least_absolute_shrinkage

Compute the coefficients of a polynomial expansion by using the LASSO problem.

**Specification**

**Alias:** lasso  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noise_tolerance</td>
<td></td>
<td></td>
<td>The noise tolerance used when performing cross validation in the presence of noise or truncation errors. The $l_2$ penalty used when performing compressed sensing with elastic net.</td>
</tr>
<tr>
<td>l2_penalty</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Compute the coefficients of a polynomial expansion by using the LASSO problem.

**noise_tolerance**

• Keywords Area
• method
• polynomial_chaos
• expansion_order
• collocation_ratio
• least_absolute_shrinkage
• noise_tolerance

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** 1e-3 for BPDN, 0. otherwise (algorithms run until termination)
6.2. METHOD

Description
The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

\texttt{l2\_penalty}

- Keywords Area
- method
- polynomial\_chaos
- expansion\_order
- collocation\_ratio
- least\_absolute\_shrinkage
- l2\_penalty

The \(l_2\) penalty used when performing compressed sensing with elastic net.

Specification
Alias: none
Argument(s): REAL
Default: 0. (reverts to standard LASSO formulation)

Description
The \(l_2\) penalty used when performing compressed sensing with elastic net.

\texttt{cross\_validation}

- Keywords Area
- method
- polynomial\_chaos
- expansion\_order
- collocation\_ratio
- cross\_validation

Use cross validation to choose the ’best’ polynomial order of a polynomial chaos expansion.

Specification
Alias: none
Argument(s): none
Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

```

use_derivatives
```

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_ratio
- use_derivatives

Use derivative data to construct surrogate models

Specification

Alias: none

- Argument(s): none
- Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

```
tensor_grid
```

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- collocation_ratio
- tensor_grid

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
6.2. **METHOD**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** regression with LHS sample set (point collocation)

**Description**

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

- `reuse_points`
  - **Keywords Area**
  - **method**
  - **polynomial_chaos**
  - **expansion_order**
  - **collocation_ratio**
  - **reuse_points**

  This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

**Specification**

**Alias:** `reuse_samples`

**Argument(s):** none

**Default:** no sample reuse in coefficient estimation

**Description**

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

- `expansion_samples`
  - **Keywords Area**
  - **method**
  - **polynomial_chaos**
  - **expansion_order**
  - **expansion_samples**

  The number of simulation samples to estimate the PCE coefficients.
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>reuse_points</td>
<td>This describes the behavior of reuse of points in constructing polynomial chaos expansion models.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>incremental_lhs</td>
<td>Augments an existing Latin Hypercube Sampling (LHS) study</td>
</tr>
</tbody>
</table>

Description

The number of simulation samples to estimate the PCE coefficients. In this case, the expansion order p cannot be inferred from the numerical integration specification, and it is necessary to provide an expansion order to specify p for a total-order expansion.

reuse_points

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- expansion_samples
- reuse_points

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

<table>
<thead>
<tr>
<th>Argument(s): none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default: no sample reuse in coefficient estimation</td>
</tr>
</tbody>
</table>

Description

The reuse_points option controls the reuse behavior of points for various types of polynomial chaos expansions, including: collocation_points, collocation_ratio, expansion_samples, or orthogonal_least_interpolation. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify reuse_points so that any points that have been previously generated (for example, from the import_points file) can be reused.
6.2. METHOD

incremental_lhs

- Keywords Area
- method
- polynomial_\text{chaos}
- expansion_order
- expansion_samples
- incremental_lhs

Augments an existing Latin Hypercube Sampling (LHS) study

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no sample reuse in coefficient estimation

**Description**

incremental\_lhs will augment an existing LHS sampling study with more samples to get better estimates of mean, variance, and percentiles. The number of samples in the second set MUST currently be 2 times the number of previous samples, although incremental sampling based on any power of two may be supported in future releases.

**Default Behavior**

Incremental Latin Hypercube Sampling is not used by default. To change this behavior, the incremental\_lhs keyword must be specified in conjuction with the sample\_type keyword. Additionally, a previous LHS (or incremental LHS) sampling study with sample size $N$ must have already been performed, and the dakota restart file must be available from this previous study. The variables and responses specifications must be the same in both studies. Incremental LHS sampling support both continuous uncertain variables and discrete uncertain variables such as discrete distributions (e.g. binomial, Poisson, etc.) as well as histogram variables and uncertain set types.

**Usage Tips**

The incremental approach is useful if it is uncertain how many simulations can be completed within available time.

See the examples below and the Usage and Restarting Dakota Studies pages.

**Examples**

For example, say a user performs an initial study using lhs as the sample\_type, and generates 10 samples.

One way to ensure the restart file is saved is to specify a non-default name, via a command line option:

```
dakota -i LHS_10.in -w LHS_10.rst
```

which uses the input file:

```
# LHS_10.in

environment
  tabular_data
    tabular_data_file = 'lhs10.dat'
```
method
   sampling
       sample_type lhs
       samples = 10

model
   single

variables
   uniform_uncertain = 2
   descriptors = ‘input1’  ‘input2’
   lower_bounds = -2.0  -2.0
   upper_bounds =  2.0   2.0

interface
   analysis_drivers ‘text_book’
   fork

responses
   response_functions = 1
   no_gradients
   no_hessians

and the restart file is written to LHS_10.rst.

Then an incremental LHS study can be run with:

dakota -i LHS_20.in -r LHS_10.rst -w LHS_20.rst

where LHS_20.in is shown below, and LHS_10.rst is the restart file containing the results of the previous LHS study.

# LHS_20.in
environment
   tabular_data
       tabular_data_file = 'lhs_incremental_20.dat'

method
   sampling
       sample_type incremental_lhs
       samples = 20
       previous_samples = 10

model
   single

variables
   uniform_uncertain = 2
   descriptors = ‘input1’  ‘input2’
   lower_bounds = -2.0  -2.0
   upper_bounds =  2.0   2.0

interface
   analysis_drivers ‘text_book’
   fork

responses
   response_functions = 1
   no_gradients
   no_hessians
The user will get 10 new LHS samples which maintain both the correlation and stratification of the original LHS sample. The new samples will be combined with the original samples to generate a combined sample of size 20.

This is clearly seen by comparing the two tabular data files.

**import_build_points_file**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **expansion_order**
- **import_build_points_file**

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file

**Argument(s):** STRING

**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:
CHAPTER 6. KEYWORDS AREA

- annotated (default)
- custom.annotated
- freeform

Examples
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
- Keywords Area
- method
- polynomial_chaos
- expansion_order
- import_build_points_file
- annotated

Selects annotated tabular file format

Topics
This keyword is related to the topics:
- file_formats

Specification
Alias: none
- Argument(s): none
- Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips
- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom.annotated
  header eval id
6.2. METHOD

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```python
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated

Resulting tabular file:

%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

- **custom_annotated**
  - **Keywords Area**
  - **method**
  - **polynomial_chaos**
  - **expansion_order**
  - **import_build_points_file**
  - **custom_annotated**

    Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- **file_formats**

Specification

**Alias:** none
**Argument(s):** none
**Default:** annotated format
### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id   x1   x2   obj_fn  nln_ineq_con_1  nln_ineq_con_2
1        0.9   1.1    0.0002   0.00026    0.76
2      0.90009 1.10 0.0001996404857 0.2601620081 0.759955
3      0.89991 1.10 0.0002003604863 0.2598380081 0.760045
...
```

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
6.2. METHOD

header

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent **custom_annotated**

**eval_id**

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent **custom_annotated**
interface_id

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- import_build_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent `custom.annotated`

`freeform`

- Keywords Area
- method
- polynomial_chaos
- expansion_order
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- `file_formats`

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format
6.2. METHOD

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform

Resulting tabular file:
```

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

- active_only

  - Keywords Area
  - method
  - polynomial_chaos
  - expansion_order
  - import_build_points_file
  - active_only
  Import only active variables from tabular data file

Topics
This keyword is related to the topics:
- file_formats
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

**orthogonal_least_interpolation**

- Keywords Area
- method
- polynomial_chaos
- orthogonal_least_interpolation

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: least_interpolation_oli

Argument(s): none

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>collocation_points</td>
<td>Specify the number of collocation points used to estimate PCE coefficients using orthogonal least interpolation</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>cross_validation</td>
<td>Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.</td>
</tr>
</tbody>
</table>
6.2. METHOD

| Optional | tensor_grid | Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion. This describes the behavior of reuse of points in constructing polynomial chaos expansion models. File containing points you wish to use to build a surrogate |
| Optional | reuse_points |
| Optional | import_build-points_file |

**Description**

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods, expansion_order cannot be set. OLI will produce the lowest degree polynomial that interpolates the data.

collocation_points
- Keywords Area
- method
- polynomial_chaos
- orthogonal_least_interpolation
- collocation_points

Specify the number of collocation points used to estimate PCE coefficients using orthogonal least interpolation.

**Specification**

Alias: none

Argument(s): INTEGERLIST

**Description**

Specify the number of collocation points used to estimate PCE coefficients using orthogonal least interpolation.

cross_validation
- Keywords Area
- method
Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

**Specification**

**Alias:** none
  **Argument(s):** none

**Description**

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

**tensor_grid**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **orthogonal_least_interpolation**
- **tensor_grid**

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

**Specification**

**Alias:** none
  **Argument(s):** INTEGERLIST
  **Default:** regression with LHS sample set (point collocation)

**Description**

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.
6.2. METHOD

reuse_points

- Keywords Area
- method
- polynomial_chaos
- orthogonal_least_interpolation
- reuse_points

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples
Argument(s): none
Default: no sample reuse in coefficient estimation

Description

The reuse_points option controls the reuse behavior of points for various types of polynomial chaos expansions, including: collocation_points, collocation_ratio, expansion_samples, or orthogonal_least_interpolation. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify reuse_points so that any points that have been previously generated (for example, from the import_points file) can be reused.

import_build_points_file

- Keywords Area
- method
- polynomial_chaos
- orthogonal_least_interpolation
- import_build_points_file

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file
Argument(s): STRING
Default: no point import from a file
### Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

### Examples

```dakota
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
  • Keywords Area
  • method
  • polynomial_chaos
  • orthogonal_least_interpolation
  • import_build_points_file
  • annotated
  Selects annotated tabular file format
```
Topics
This keyword is related to the topics:

- file_formats

Specification

Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
custom.annotated

- Keywords Area
- method
- polynomial.choas
- orthogonal.least.interpolation
- import.build.points.file
- custom.annotated

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- fileformats

**Specification**

Alias: none  
Argument(s): none  
Default: annotated format

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>header</td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td>eval_id</td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td>interface_id</td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**
The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

header

- Keywords Area
- method
- polynomial_chaos
- orthogonal_least_interpolation
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated
Eval_id

- Keywords Area
- Method
- Polynomial_chaos
- Orthogonal_least_interpolation
- Import_build_points_file
- Custom_annotated
- Eval_id

Enable evaluation ID column in custom-annotated tabular file

### Specification

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

### Description

See description of parent custom_annotated

Interface_id

- Keywords Area
- Method
- Polynomial_chaos
- Orthogonal_least_interpolation
- Import_build_points_file
- Custom_annotated
- Interface_id

Enable interface ID column in custom-annotated tabular file

### Specification

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

### Description

See description of parent custom_annotated
6.2. METHOD

freeform

• Keywords Area
• method
• polynomial_chaos
• orthogonal_least_interpolation
• import_build_points_file
• freeform

Selects freeform file format

Topics
This keyword is related to the topics:

• file_formats

Specification

Alias: none
  Argument(s): none
  Default: annotated format

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```python
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
  0.9   1.1   0.0002   0.26   0.76
  0.90009   1.1   0.0001996404857   0.2601620081   0.759955
  0.89991   1.1   0.0002003604863   0.2598380081   0.760045
...
```

 active_only

- Keywords Area
- method
- polynomial_chaos
- orthogonal_least_interpolation
- import_build_points_file
- active_only

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword active_only indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.
6.2. METHOD

import_expansion_file
- Keywords Area
- method
- polynomial_chaos
- import_expansion_file

Build a Polynomial Chaos Expansion (PCE) by import coefficients and a multi-index from a file

Specification
Alias: none
Argument(s): STRING

Description
The coefficients can be specified in an arbitrary order. The multi-index provided is used to generate a sparse expansion that consists only of the indices corresponding to the non-zero coefficients provided in the file.

variance_based_decomp
- Keywords Area
- method
- polynomial_chaos
- variance_based_decomp

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification
Alias: none
Argument(s): none
Default: no variance-based decomposition

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>interaction_order</td>
<td></td>
<td>Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.</td>
</tr>
</tbody>
</table>
**CHAPTER 6. KEYWORDS AREA**

---

**Optional**

| drop_tolerance | Suppresses output of sensitivity indices with values lower than this tolerance |

---

**Description**

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

**Default Behavior**

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol’ indices.

**Expected Outputs**

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of univariate terms for each individual variable, interaction effects represent the set of mixed terms (the complement of the univariate set), and total effects represent the complete set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

**Usage Tips**

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

**Examples**

```plaintext
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

**Theory**

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of[73]: “The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input.”

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, $S_i$, means that the uncertainty in the input variable $i$ has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in[87].
6.2. METHOD

- method
- polynomial_chaos
- variance_based_decomp
- interaction_order

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** Unrestricted (VBD includes all interaction orders present in the expansion)

**Description**

The interaction_order option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous univariate_effects specification is equivalent to interaction_order = 1 in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume).

**drop_tolerance**

- Keywords Area  
- method  
- polynomial_chaos  
- variance_based_decomp  
- drop_tolerance

suppresses output of sensitivity indices with values lower than this tolerance

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** All VBD indices displayed

**Description**

The drop_tolerance keyword allows the user to specify a value below which sensitivity indices generated by variance_based_decomp are not displayed.

**Default Behavior**

By default, all sensitivity indices generated by variance_based_decomp are displayed.

**Usage Tips**
For polynomial_chaos, which outputs main, interaction, and total effects by default, the univariate-effects may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```plaintext
method,
sampling
  sampling_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

diagonal_covariance

- Keywords Area
- method
- polynomial_chaos
- diagonal_covariance

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: diagonal_covariance for response vector > 10; else full_covariance

Description

With a large number of responses, the covariance matrix can be very large. diagonal_covariance is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

full_covariance

- Keywords Area
- method
- polynomial_chaos
- full_covariance

Display the full covariance matrix

Specification

Alias: none

Argument(s): none
Description

With a large number of responses, the covariance matrix can be very large. \texttt{full\_covariance} is used to force Dakota to output the full covariance matrix.

normalized

- Keywords Area
- method
- polynomial\_chaos
- normalized

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials.

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials.

sample\_type

- Keywords Area
- method
- polynomial\_chaos
- sample\_type

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: lhs

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required (Choose One)</th>
<th>Group 1</th>
<th>Group 2</th>
<th>Uses Latin Hypercube Sampling (LHS) to sample variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>lhs</td>
<td>random</td>
<td>random</td>
<td>Uses purely random Monte Carlo sampling to sample variables</td>
</tr>
</tbody>
</table>

**Description**

The `sample_type` keyword allows the user to select between multiple random sampling approaches. There are two primary types of sampling: Monte Carlo (pure random) and Latin Hypercube Sampling. Additionally, these methods have incremental variants that allow an existing study to be augmented with additional samples to get better estimates of mean, variance, and percentiles.

**Default Behavior**

If the `sample_type` keyword is present, it must be accompanied by `lhs`, `random`, `incremental_lhs`, or `incremental_random`. Otherwise, `lhs` will be used by default.

**Examples**

```plaintext
method sampling
    sample_type lhs
    samples = 20
```

**lhs**

- Keywords Area
- method
- polynomial_chaos
- sample_type
- lhs

Uses Latin Hypercube Sampling (LHS) to sample variables

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

**Default Behavior**
6.2. METHOD

By default, Latin Hypercube Sampling is used. To explicitly specify this in the Dakota input file, however, the \texttt{lhs} keyword must appear in conjunction with the \texttt{sample_type} keyword.

**Usage Tips**

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 20
```

**random**

- **Keywords Area**
- **method**
- **polynomial\_chaos**
- **sample\_type**
- **random**

Uses purely random Monte Carlo sampling to sample variables

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The \texttt{random} keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

**Default Behavior**

Monte Carlo sampling is not used by default. To change this behavior, the \texttt{random} keyword must be specified in conjunction with the \texttt{sample\_type} keyword.

**Usage Tips**

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

**Examples**

```plaintext
method
  sampling
    sample_type random
    samples = 200
```
probability_refinement

- Keywords Area
- method
- polynomial_chaos
- probability_refinement

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- reliability_methods

Specification

Alias: sample_refinement

Argument(s): none

Default: no refinement

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>import</td>
<td>Sampling option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adapt_import</td>
<td>Importance sampling option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mm_adapt_import</td>
<td>Sampling option</td>
</tr>
<tr>
<td>Optional</td>
<td>refinement_samples</td>
<td></td>
<td>Specify the number of samples used to improve a probability estimate.</td>
</tr>
</tbody>
</table>

Description

The probability_refinement allows refinement of probability and generalized reliability results using importance sampling. If one specifies probability_refinement, there are some additional options. One can specify which type of importance sampling to use (import, adapt_import, or mm_adapt_import). Additionally, one can specify the number of refinement samples to use with refinement_samples and the seed to use with seed.

The probability_refinement density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

import

- Keywords Area
- method
- polynomial_chaos
6.2. METHOD

- probability_refinement
- import

Sampling option

Specification

Alias: none
  Argument(s): none

Description

import centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

adapt_import

- Keywords Area
- method
- polynomial_chaos
- probability_refinement
- adapt_import

Importance sampling option

Specification

Alias: none
  Argument(s): none

Description

adapt_import centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

mm_adapt_import

- Keywords Area
- method
- polynomial_chaos
- probability_refinement
- mm_adapt_import

Sampling option
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

`refinement_samples`

- Keywords Area
- method
- polynomial_chaos
- probability_refinement
- refinement_samples

Specify the number of samples used to improve a probability estimate.

Specification

Alias: none
Argument(s): INTEGER

Description

Specify the number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

`import_approx_points_file`

- Keywords Area
- method
- polynomial_chaos
- `import_approx_points_file`

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none
Argument(s): STRING
Default: no point import from a file
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

Description

**Default Behavior**  No import of points at which to evaluate the surrogate.

**Expected Output**  The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```plaintext
method
  polynomial_chaos
  expansion_order = 4
  import_approx_points_file = 'import.mcmc_annot.dat'
  annotated
```

annotated

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **import_approx_points_file**
- **annotated**

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- **file_formats**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none
Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.9009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

custom_annotated

- Keywords Area
- method
- polynomial_chaos
- import_approx_points_file
- custom_annotated

Selects custom-annotated tabular file format
6.2. METHOD

Topics
This keyword is related to the topics:

- file_formats

Specification

**Alias:** none  
**Argument(s):** none  
**Default:** annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom.annotated, followed by options, in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id

Resulting tabular file:

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

definition
  description
    keywords_area
      method
      polynomial_chaos
      import_approx_points_file
      custom_annotated
      header

Enable header row in custom-annotated tabular file

Specification

Alias: none
  Argument(s): none
  Default: no header

Description

See description of parent custom_annotated

eval_id
  keywords_area
  method
  polynomial_chaos
  import_approx_points_file
  custom_annotated
  eval_id

Enable evaluation ID column in custom-annotated tabular file
### 6.2. METHOD

**Specification**

**Alias:** none
- **Argument(s):** none
- **Default:** no eval_id column

**Description**

See description of parent `custom.annotated`

```plaintext
interface_id
```
- **Keywords Area**
- **method**
- **polynomial_chaos**
- **import_approx_points_file**
- **custom.annotated**
- **interface_id**

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none
- **Argument(s):** none
- **Default:** no interface_id column

**Description**

See description of parent `custom.annotated`

```plaintext
freeform
```
- **Keywords Area**
- **method**
- **polynomial_chaos**
- **import_approx_points_file**
- **freeform**

Selects freeform file format

**Topics**

This keyword is related to the topics:
- **file_formats**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9   1.1   0.0002   0.26   0.76
0.00009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

active_only

• Keywords Area

• method

• polynomial_chaos

• import_approx_points_file

• active_only

Import only active variables from tabular data file
6.2. METHOD

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword active_only indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

export_approx_points_file

- Keywords Area
- method
- polynomial_chaos
- export_approx_points_file

Output file for evaluations of a surrogate model

Specification

Alias: export_points_file

Argument(s): STRING

Default: no point export to a file

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| freeform | Selects freeform file format |

**Description**

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.

**Usage Tips**

Dakota exports tabular data in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**annotated**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **export_approx_points_file**
- annotated

Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

- **file_formats**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format

**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

**Usage Tips**
6.2. METHOD

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```python
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

custom_annotated

- Keywords Area
- method
- polynomial_chaos
- export_approx_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format
### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
6.2. METHOD

header

- Keywords Area
- method
- polynomial_chaos
- export_approx_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

- **Argument(s):** none
- **Default:** no header

**Description**

See description of parent custom_annotated

```
 eval_id
```

- Keywords Area
- method
- polynomial_chaos
- export_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

- **Argument(s):** none
- **Default:** no eval_id column

**Description**

See description of parent custom_annotated
interface_id

- Keywords Area
- method
- polynomial_chaos
- export_approx_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent custom.annotated

freeform

- Keywords Area
- method
- polynomial_chaos
- export_approx_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format
6.2. METHOD

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

```
export_expansion_file
  Keywords Area
  method
  polynomial_chaos
  export_expansion_file
```

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification
Alias: none
Argument(s): STRING
Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consists only of the indices corresponding to the non-zero coefficients of the PCE.

**fixed_seed**
- **Keywords Area**
- **method**
- **polynomial_chaos**
- **fixed_seed**

Reuses the same seed value for multiple random sampling sets

Specification

**Alias:** none
**Argument(s):** none
**Default:** not fixed; pattern varies run-to-run

Description

The **fixed_seed** flag is relevant if multiple sampling sets will be generated over the coarse of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

**Default Behavior**

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the **seed** must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
  fixed_seed
```

**max_iterations**
- **Keywords Area**
- **method**
- **polynomial_chaos**
- **max_iterations**

Stopping criterion based on number of iterations
6.2. METHOD

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none

Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global.{reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- polynomial_chaos
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:

- method_independent_controls

Specification
Alias: none

Argument(s): REAL
Default: 1.e-4

Description
The convergence_tolerance specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library
This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.
Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

**reliability_levels**

- Keywords Area
- method
- polynomial_chaos
- reliability_levels

Specify reliability levels at which the response values will be estimated

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_reliability_levels</td>
<td></td>
<td>Specify which reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

**Theory**

Sets of response-probability pairs computed with the forward/reverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.
An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_reliability_levels**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **reliability_levels**
- **num_reliability_levels**

Specify which reliability_levels correspond to which response

**Specification**

Alias: none

Argument(s): INTEGERLIST

Default: reliability_levels evenly distributed among response functions

**Description**

See parent page

**response_levels**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **response_levels**

Values at which to estimate desired statistics for each response

**Specification**

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>num_response_-levels</td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>compute</td>
<td>Selection of statistics to compute at each response level</td>
</tr>
</tbody>
</table>

Description

The response_levels specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If response_levels are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If response_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The num.response_levels is used to specify which arguments of the response.level correspond to which response.

Examples

For example, specifying a response.level of 52.3 followed with compute probabilities will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```plaintext
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the num_response_levels key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```plaintext
method, sampling,
samples = 100 seed = 1
```
6.2. METHOD

complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
6.0e+04 6.5e+04 7.0e+04
3.5e+05 4.0e+05 4.5e+05

variables,
normal_uncertain = 2
means = 248.89, 593.33
std_deviations = 12.4, 29.7
descriptors = 'TF1n' 'TF2n'
uniform_uncertain = 2
lower_bounds = 199.3, 474.63
upper_bounds = 298.5, 712.
descriptors = 'TF1u' 'TF2u'
weibull_uncertain = 2
alphas = 12., 30.
betas = 250., 590.
descriptors = 'TF1w' 'TF2w'
histogram_bin_uncertain
num_pairs = 3 4
abscissas = 5 8 10 .1 .2 .3 .4
counts = 17 21 0 12 24 12 0
descriptors = 'TF1h' 'TF2h'
histogram_point_uncertain
real = 1
num_pairs = 2
abscissas = 3 4
counts = 1 1
descriptors = 'TF3h'

interface,
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses,
response_functions = 3
no_gradients
no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.034186485e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

3.5000000000e+05  4.0000000000e+05  8.6000000000e-06
4.0000000000e+05  4.5000000000e+05  1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

num_response_levels

- Keywords Area
- method
- polynomial_chaos
- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions
6.2. METHOD

Description
The num_response_levels keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior
If num_response_levels is not specified, the response_levels will be evenly distributed among the responses.

Expected Outputs
The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples
```
method sampling
  samples = 100
  seed = 34785
  num_response_levels = 1 1 1
  response_levels = 0.5 0.5 0.5
```

compute
- Keywords Area
- method
- polynomial_chaos
- response_levels
- compute

Selection of statistics to compute at each response level

Specification
Alias: none
Argument(s): none
Default: probabilities

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Probabilities</td>
<td>Group 1</td>
<td>Computes probabilities associated with response levels</td>
</tr>
</tbody>
</table>
**CHAPTER 6. KEYWORDS AREA**

<table>
<thead>
<tr>
<th>compute keyword</th>
<th>reliabilities</th>
<th>Computes reliabilities associated with response levels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gen_reliabilities</td>
<td>Computes generalized reliabilities associated with response levels</td>
</tr>
</tbody>
</table>

**Optional**

<table>
<thead>
<tr>
<th>system</th>
<th>Compute system reliability (series or parallel)</th>
</tr>
</thead>
</table>

**Description**

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

**Default Behavior**

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

**Expected Output**

The type of statistics specified by `compute` will be reported for each response level.

**Usage Tips**

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach. CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

**Examples**

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11 6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
  probabilities
  
  Keywords Area
  method
  polynomial_chaos
  response_levels
```
6.2. METHOD

- compute
- probabilities

Computes probabilities associated with response levels

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

**Examples**

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
    6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
    compute probabilities

reliabilities
```

**Specification**

**Alias:** none

**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description

The **reliabilities** keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If **response_levels** is specified, the reliabilities are not computed by default. To change this behavior, the **reliabilities** keyword should be specified in conjunction with the **compute** keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```plaintext
method sampling
  sample_type random
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
  compute reliabilities
```

**gen_reliabilities**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **response_levels**
- **compute**
- **gen_reliabilities**

Computes generalized reliabilities associated with response levels

Specification

Alias: none  
Argument(s): none

Description

The **gen_reliabilities** keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If **response_levels** is specified, the generalized reliabilities are not computed by default. To change this behavior, the **gen_reliabilities** keyword should be specified in conjunction with the **compute** keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.
6.2. METHOD

Examples

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities

system

• Keywords Area
• method
• polynomial_chaos
• response_levels
• compute
• system

Compute system reliability (series or parallel)
```

Specification

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>series</td>
<td>Aggregate response statistics assuming a series system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parallel</td>
<td>Aggregate response statistics assuming a parallel system</td>
</tr>
</tbody>
</table>

**Description**

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.
series
- Keywords Area
- method
- polynomial_chaos
- response_levels
- compute
- system
- series

Aggregate response statistics assuming a series system

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

See parent keyword `system` for description.

parallel
- Keywords Area
- method
- polynomial_chaos
- response_levels
- compute
- system
- parallel

Aggregate response statistics assuming a parallel system

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

See parent keyword `system` for description.
6.2. METHOD

distribution

- Keywords Area
- method
- polynomial_chaos
- distribution

Selection of cumulative or complementary cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>

**Description**

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

**Default Behavior**

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

**Examples**

```
method sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```
cumulative

- Keywords Area
- method
- polynomial_chaos
- distribution
- cumulative

Computes statistics according to cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.  

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the *cumulative* keyword must be appear in conjunction with the *distribution* keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

complementary

- Keywords Area
- method
- polynomial_chaos
- distribution
- complementary

Computes statistics according to complementary cumulative functions
6.2. *METHOD*

**Specification**

*Alias:* none  
*Argument(s):* none

**Description**

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

**Examples**

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

**probability_levels**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **probability_levels**

Specify probability levels at which to estimate the corresponding response value

**Specification**

*Alias:* none  
*Argument(s):* REALLIST  
*Default:* No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>num_probability_levels</td>
</tr>
</tbody>
</table>

Specify which probability levels correspond to which response
CHAPTER 6. KEYWORDS AREA

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method, sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = 'TF1w' 'TF2w'

histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = 'TF1h' 'TF2h'

histogram_point_uncertain
  real = 1
  num_pairs = 2
  abscissas = 3 4
  counts = 1 1
  descriptors = 'TF3h'

interface,
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```
Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0634975300e+11</td>
<td>5.145122311e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.4196114379e+11</td>
<td>2.433429039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.643154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.379637090e+05</td>
<td>3.3000000000e+05</td>
<td>5.3028386523e-06</td>
</tr>
<tr>
<td>3.3000000000e+05</td>
<td>3.8100966235e+05</td>
<td>9.0600056364e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.411498127e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.0000000000e+00</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.0000000000e-01</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>3.0000000000e-01</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.643154744e+04</td>
<td>1.0000000000e+00</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.0000000000e-01</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>5.0000000000e-01</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.379637090e+05</td>
<td>1.0000000000e+00</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>3.3000000000e+05</td>
<td>3.0000000000e-01</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>2.0000000000e-01</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>4.411498127e+05</td>
<td>0.0000000000e+00</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).
Keywords Area

- method
- polynomial_chaos
- probability_levels
- num_probability_levels

Specify which probability_levels correspond to which response

**Specification**

**Alias:** none  
**Argument(s):** INTEGERLIST  
**Default:** probability_levels evenly distributed among response functions

**Description**

See parent page

gen_reliability_levels

- Keywords Area
- method
- polynomial_chaos
- gen_reliability_levels

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_gen_reliability_levels</td>
<td></td>
<td>Specify which gen_reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).
6.2. METHOD

Theory
Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_gen_reliability_levels

- Keywords Area
- method
- polynomial_chaos
- gen_reliability_levels
- num_gen_reliability_levels

Specify which gen_reliability_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability_levels evenly distributed among response functions

Description

See parent page

rng

- Keywords Area
- method
- polynomial_chaos
- rng

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister ( mt19937 )
### Keywords Area

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
<tr>
<td>rnum2</td>
<td>Generates pseudo-random numbers using the Pecos package</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Description

The `rng` keyword is used to indicate a choice of random number generator.

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**

The default is recommended, as the Mersenne twister is a higher quality random number generator.

### Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

**mt19937**

- **Keywords Area**
- **method**
- **polynomial_chaos**
- **rng**
- **mt19937**

Generates random numbers using the Mersenne twister

### Specification

**Alias:** none  
**Argument(s):** none
6.2. METHOD

Description

The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on Wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

Default Behavior

mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended.

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

rnum2

- Keywords Area
- method
- polynomial_chaos
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```
samples

- Keywords Area
- method
- polynomial_chaos
- samples

Number of samples for sampling-based methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least \( \text{dim+1} \) samples should be used, where “\( \text{dim} \)” is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least \( (\text{dim+1})(\text{dim+2})/2 \) samples are needed. For uncertainty quantification, we recommend at least \( 10^*\text{dim} \) samples. For variance_based_decomp, we recommend hundreds to thousands of samples. Note that for variance_based_decomp, the number of simulations performed will be \( N*(\text{dim+2}) \).

**Examples**

```dakota
method
  sampling
    sample_type lhs
    samples = 20
```

**seed**

- Keywords Area
- method
- polynomial_chaos
- seed

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)
6.2. METHOD

Description
The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 1534?
```

model_pointer

- Keywords Area
- method
- polynomial_choas
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.
Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.47 stoch_collocation

  • Keywords Area
### 6.2. METHOD

- method
- stoch_collocation

Uncertainty quantification with stochastic collocation

**Specification**

**Alias:** nond_stoch_collocation  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>Automated refinement type (Group 1)</td>
<td>p_refinement</td>
<td>Automatic polynomial order refinement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h_refinement</td>
<td>Employ h-refinement to refine the grid</td>
</tr>
<tr>
<td><strong>Optional (Choose One)</strong></td>
<td>Basis polynomial family (Group 2)</td>
<td>piecewise</td>
<td>Use piecewise local basis functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>askey</td>
<td>Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wiener</td>
<td>Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.</td>
</tr>
<tr>
<td>Required <em>(Choose One)</em></td>
<td>Interpolation grid type (Group 3)</td>
<td>\texttt{quadrature_order}</td>
<td>Cubature using tensor-products of Gaussian quadrature rules</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{sparse_grid_level}</td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>\texttt{dimension_preference}</td>
<td>A set of weights specifying the relative importance of each uncertain variable (dimension)</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>\texttt{use_derivatives}</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
<tr>
<td>Optional <em>(Choose One)</em></td>
<td>Nesting of quadrature rules (Group 4)</td>
<td>\texttt{nested}</td>
<td>Enforce use of nested quadrature rules if available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{non_nested}</td>
<td>Enforce use of non-nested quadrature rules</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>\texttt{variance_based_decomp}</td>
<td>Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects</td>
</tr>
</tbody>
</table>
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>Covariance type (Group 5)</th>
<th>diagonal covariance</th>
<th>full covariance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>sample_type</td>
<td>Display only the diagonal terms of the covariance matrix</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>probability_refinement</td>
<td>Display the full covariance matrix</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>import_approx_points_file</td>
<td>Selection of sampling strategy</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>export_approx_points_file</td>
<td>Allow refinement of probability and generalized reliability results using importance sampling</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>fixed_seed</td>
<td>Filename for points at which to evaluate the PCE/SC surrogate</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Output file for evaluations of a surrogate model</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td>Reuses the same seed value for multiple random sampling sets</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
</tbody>
</table>
### Optional

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reliability_levels</code></td>
<td>Specify reliability levels at which the response values will be estimated.</td>
</tr>
<tr>
<td><code>response_levels</code></td>
<td>Values at which to estimate desired statistics for each response.</td>
</tr>
<tr>
<td><code>distribution</code></td>
<td>Selection of cumulative or complementary cumulative functions.</td>
</tr>
<tr>
<td><code>probability_levels</code></td>
<td>Specify probability levels at which to estimate the corresponding response value.</td>
</tr>
<tr>
<td><code>gen_reliability_levels</code></td>
<td>Specify generalized reliability levels at which to estimate the corresponding response value.</td>
</tr>
<tr>
<td><code>rng</code></td>
<td>Selection of a random number generator.</td>
</tr>
<tr>
<td><code>samples</code></td>
<td>Number of samples for sampling-based methods.</td>
</tr>
<tr>
<td><code>seed</code></td>
<td>Seed of the random number generator.</td>
</tr>
<tr>
<td><code>model_pointer</code></td>
<td>Identifier for model block to be used by a method.</td>
</tr>
</tbody>
</table>

### Description

Stochastic collocation is a general framework for approximate representation of random response functions in terms of finite-dimensional interpolation bases.

The stochastic collocation (SC) method is very similar to polynomial_chaos, with the key difference that the orthogonal polynomial basis functions are replaced with interpolation polynomial bases. The interpolation polynomials may be either local or global and either value-based or gradient-enhanced. In the local case, valued-
6.2. METHOD

Based are piecewise linear splines and gradient-enhanced are piecewise cubic splines, and in the global case, valued-based are Lagrange interpolants and gradient-enhanced are Hermite interpolants. A value-based expansion takes the form

$$ R = \sum_{i=1}^{N_p} r_i L_i(\xi) $$

where $N_p$ is the total number of collocation points, $r_i$ is a response value at the $i^{th}$ collocation point, $L_i$ is the $i^{th}$ multidimensional interpolation polynomial, and $\xi$ is a vector of standardized random variables.

Thus, in PCE, one forms coefficients for known orthogonal polynomial basis functions, whereas SC forms multidimensional interpolation functions for known coefficients.

**Basis polynomial family (Group 2)**

In addition to the askey and wiener basis types also supported by polynomial chaos, SC supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the use_derivatives specification, piecewise cubic Hermite splines. Both of these basis options provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space. The global gradient-enhanced interpolants (Hermite interpolation polynomials) are also restricted to uniform or transformed uniform random variables (due to the need to compute collocation weights by integration of the basis polynomials) and share the variable support shown in variable support for Piecewise SE. Due to numerical instability in these high-order basis polynomials, they are deactivated by default but can be activated by developers using a compile-time switch.

**Interpolation grid type (Group 3)**

To form the multidimensional interpolants $L_i$ of the expansion, two options are provided.

1. interpolation on a tensor-product of Gaussian quadrature points (specified with quadrature_order and, optionally, dimension_preference for anisotropic tensor grids). As for PCE, non-nested Gauss rules are employed by default, although the presence of p_refinement or h_refinement will result in default usage of nested rules for normal or uniform variables after any variable transformations have been applied (both defaults can be overridden using explicit nested or non_nested specifications).

2. interpolation on a Smolyak sparse grid (specified with sparse_grid_level and, optionally, dimension_preference for anisotropic sparse grids) defined from Gaussian rules. As for sparse PCE, nested rules are employed unless overridden with the non_nested option, and the growth rules are restricted unless overridden by the unrestricted keyword.

Another distinguishing characteristic of stochastic collocation relative to polynomial_chaos is the ability to reformulate the interpolation problem from a nodal interpolation approach into a hierarchical formulation in which each new level of interpolation defines a set of incremental refinements (known as hierarchical surpluses) layered on top of the interpolants from previous levels. This formulation lends itself naturally to uniform or adaptive refinement strategies, since the hierarchical surpluses can be interpreted as error estimates for the interpolant. Either global or local/piecewise interpolants in either value-based or gradient-enhanced approaches can be formulated using hierarchical interpolation. The primary restriction for the hierarchical case is that it currently requires a sparse grid approach using nested quadrature rules (Genz-Keister, Gauss-Patterson, or Newton-Cotes for standard normals and standard uniforms in a transformed space: Askey, Wiener, or Piecewise settings may be required), although this restriction can be relaxed in the future. A selection of hierarchical interpolation
CHAPTER 6. KEYWORDS AREA

will provide greater precision in the increments to mean, standard deviation, covariance, and reliability-based level mappings induced by a grid change within uniform or goal-oriented adaptive refinement approaches (see following section).

It is important to note that, while quadrature order and sparse_grid_level are array inputs, only one scalar from these arrays is active at a time for a particular expansion estimation. These scalars can be augmented with a dimension_preference to support anisotropy across the random dimension set. The array inputs are present to support advanced use cases such as multifidelity UQ, where multiple grid resolutions can be employed.

Automated refinement type (Group 1)

Automated expansion refinement can be selected as either p_refinement or h_refinement, and either refinement specification can be either uniform or dimension adaptive. The dimension adaptive case can be further specified as either sobol or generalized (decay not supported). Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls. The h_refinement specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify piecewise in the case of h_refinement). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current uniform and dimension_adaptive options.

Covariance type (Group 5)

These two keywords are used to specify how this method computes, stores, and outputs the covariance of the responses. In particular, the diagonal covariance option is provided for reducing post-processing overhead and output volume in high dimensional applications.

Active Variables

The default behavior is to form expansions over aleatory uncertain continuous variables. To form expansions over a broader set of variables, one needs to specify active followed by state, epistemic, design, or all in the variables specification block.

For continuous design, continuous state, and continuous epistemic uncertain variables included in the expansion, interpolation points for these dimensions are based on Gauss-Legendre rules if non-nested, Gauss-Patterson rules if nested, and Newton-Cotes points in the case of piecewise bases. Again, when probability integrals are evaluated, only the aleatory random variable domain is integrated, leaving behind a polynomial relationship between the statistics and the remaining design/state/epistemic variables.

Optional Keywords regarding method outputs

Each of these sampling specifications refer to sampling on the SC approximation for the purposes of generating approximate statistics.

- sample_type
- samples
- seed
- fixed_seed
- rng
- probability_refinement
- distribution
- reliability_levels
- response_levels
- probability_levels
6.2. METHOD

- **gen.reliability_levels**

Since SC approximations are formed on structured grids, there should be no ambiguity with simulation sampling for generating the SC expansion.

When using the `probability.refinement` control, the number of refinement samples is not under the user’s control (these evaluations are approximation-based, so management of this expense is less critical). This option allows for refinement of probability and generalized reliability results using importance sampling.

**Multi-fidelity UQ**

When using multifidelity UQ, the high fidelity expansion generated from combining the low fidelity and discrepancy expansions retains the polynomial form of the low fidelity expansion (only the coefficients are updated). Refer to `polynomial_chaos` for information on the multifidelity interpretation of array inputs for `quadrature_order` and `sparse_grid_level`.

**Usage Tips**

If \( n \) is small, then tensor-product Gaussian quadrature is again the preferred choice. For larger \( n \), tensor-product quadrature quickly becomes too expensive and the sparse grid approach is preferred. For self-consistency in growth rates, nested rules employ restricted exponential growth (with the exception of the `dimension._adaptive_p.refinement generalized` case) for consistency with the linear growth used for non-nested Gauss rules (integrand precision \( i = 4l + 1 \) for sparse grid level \( l \) and \( i = 2m - 1 \) for tensor grid order \( m \)).

**Additional Resources**


**Examples**

```plaintext
method,
    stoch_collocation
    sparse_grid_level = 2
    samples = 10000 seed = 12347 rng rnum2
    response_levels = .1 1. 50. 100. 500. 1000.
    variance_based_decomp
```

**Theory**

As mentioned above, a value-based expansion takes the form

\[
R = \sum_{i=1}^{N_p} r_i L_i(\xi)
\]

The \( i^{th} \) interpolation polynomial assumes the value of 1 at the \( i^{th} \) collocation point and 0 at all other collocation points, involving either a global Lagrange polynomial basis or local piecewise splines. It is easy to see that the approximation reproduces the response values at the collocation points and interpolates between these values at other points. A gradient-enhanced expansion (selected via the `use_derivatives` keyword) involves both type 1 and type 2 basis functions as follows:

\[
R = \sum_{i=1}^{N_p} [r_i H_i^{(1)}(\xi) + \sum_{j=1}^{n} \frac{dr_i}{d\xi_j} H_i^{(2)}(\xi)]
\]

where the \( i^{th} \) type 1 interpolant produces 1 for the value at the \( i^{th} \) collocation point, 0 for values at all other collocation points, and 0 for derivatives (when differentiated) at all collocation points, and the \( ij^{th} \) type 2 interpolant produces 0 for values at all collocation points, 1 for the \( j^{th} \) derivative component at the \( i^{th} \) collocation point.
point, and 0 for the \( j^{th} \) derivative component at all other collocation points. Again, this expansion reproduces the response values at each of the collocation points, and when differentiated, also reproduces each component of the gradient at each of the collocation points. Since this technique includes the derivative interpolation explicitly, it eliminates issues with matrix ill-conditioning that can occur in the gradient-enhanced PCE approach based on regression. However, the calculation of high-order global polynomials with the desired interpolation properties can be similarly numerically challenging such that the use of local cubic splines is recommended due to numerical stability.

**See Also**

These keywords may also be of interest:

- adaptive_sampling
- gpais
- local_reliability
- global_reliability
- sampling
- importance_sampling
- polynomial_chaos

**p_refinement**

- Keywords Area
- method
- stoch_collocation
- p_refinement

Automatic polynomial order refinement

**Specification**

Alias: none

**Argument(s):** none

**Default:** no refinement

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group p-refinement type (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uniform</td>
<td></td>
<td>Refine an expansion uniformly in all dimensions.</td>
</tr>
</tbody>
</table>
6.2. **METHOD**

| dimension_adaptive | Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher ‘importance’. |

**Description**

The `prefinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol’ indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User’s Manual PCE descriptions for additional specifics).

For the case of `prefinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

**uniform**

- **Keywords Area**
- **method**
- **stoch_collocation**
• p_refinement
• uniform

Refine an expansion uniformly in all dimensions.

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

**dimension_adaptive**

• Keywords Area
• method
• stoch_collocation
• p_refinement
• dimension_adaptive

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher ‘importance’.

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> (Choose One)</td>
<td>dimension adaptivity estimation approach (Group 1)</td>
<td>sobol</td>
<td>Estimate dimension preference for automated refinement of stochastic expansion using total Sobol’ sensitivity indices.</td>
</tr>
</tbody>
</table>
6.2. METHOD

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Table

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher ‘importance’ in resolving statistical quantities of interest. Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.</td>
</tr>
</tbody>
</table>

**sobol**
- Keywords Area
- method
- stoch_collocation
- p_refinement
- dimension_adaptive
- sobol

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol’ sensitivity indices.

**Specification**

Alias: none

Argument(s): none

Default: generalized

**Description**

Determine dimension preference for refinement of a stochastic expansion from the total Sobol’ sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

**Examples**

```plaintext
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```
Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

**Specification**

*Alias:* none  
*Argument(s):* none

**Description**

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User’s Manual PCE descriptions for additional specifics).

**Examples**

```plaintext
method, polynomial_chaos
  sparse_grid_level = 3
dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

**h_refinement**

*Alias:* none  
*Argument(s):* none

*Default:* no refinement

Employ h-refinement to refine the grid

**Specification**

*Alias:* none  
*Argument(s):* none

*Default:* no refinement
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>h-refinement type (Group 1)</td>
<td>uniform</td>
<td>Refine an expansion uniformly in all dimensions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dimension_adaptive</td>
<td>Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher ‘importance’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>local_adaptive</td>
<td>Planned future capability for local pointwise refinement within a generalized sparse grid.</td>
</tr>
</tbody>
</table>

**Description**

Automated expansion refinement can be selected as either p_refinement or h_refinement, and either refinement specification can be either uniform or dimension_adaptive. The dimension_adaptive case can be further specified as either sobol or generalized (decay not supported). Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls. The h_refinement specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify piecewise in the case of h_refinement). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current uniform and dimension_adaptive options.

**uniform**

- **Keywords Area**
- method
- stoch_collocation
- h_refinement
- uniform

Refine an expansion uniformly in all dimensions.
Specification

Alias: none
  Argument(s): none

Description

The quadrature order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

dimension_adaptive

- Keywords Area
- method
- stoch_collocation
- h_refinement
- dimension_adaptive

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher ‘importance’.

Specification

Alias: none
  Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required(Choose One)</th>
<th>Description of Group dimension adaptivity estimation approach (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dakota Keyword</td>
<td>Sobol</td>
<td>Estimate dimension preference for automated refinement of stochastic expansion using total Sobol’ sensitivity indices.</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher ‘importance’ in resolving statistical quantities of interest. Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.</td>
</tr>
</tbody>
</table>

**sobol**
- Keywords Area
- method
- stoch_collocation
- h_refinement
- dimension_adaptive
- sobol

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol’ sensitivity indices.

**Specification**

Alias: none
**Argument(s):** none
**Default:** generalized

**Description**

Determine dimension preference for refinement of a stochastic expansion from the total Sobol’ sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

**Examples**

```
method,
    polynomial_chaos
    sparse_grid_level = 3
    dimension_adaptive p_refinement sobol
    max_iterations = 20
    convergence_tol = 1.e-4
```
CHAPTER 6. KEYWORDS AREA

generalized

- Keywords Area
- method
- stoch_collocation
- h.refinement
- dimension_adaptive
- generalized

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none
Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User’s Manual PCE descriptions for additional specifics).

Examples

```plaintext
method, polynomial_chaos
sparse_grid_level = 3
dimension_adaptive p_refinement generalized
max_iterations = 20
convergence_tol = 1.e-4
```

local_adaptive

- Keywords Area
- method
- stoch_collocation
- h.refinement
- local_adaptive

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none
Argument(s): none
6.2. METHOD

Description
Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

piecewise

- Keywords Area
- method
- stoch_collocation
- piecewise

Use piecewise local basis functions

Specification
Alias: none
Argument(s): none
Default: extended (Askey + numerically-generated)

Description
SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the use_derivatives specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

askey

- Keywords Area
- method
- stoch_collocation
- askey

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification
Alias: none
Argument(s): none
Default: extended (Askey + numerically-generated)
**Description**

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

**See Also**

These keywords may also be of interest:

- *polynomial_chaos*
- *wiener*

**wiener**

- *Keywords Area*
- *method*
- *stoch_collocation*
- *wiener*

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** extended (Askey + numerically-generated)

**Description**

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

**See Also**

These keywords may also be of interest:

- *polynomial_chaos*
- *askey*
6.2. METHOD

quadrature_order

- Keywords Area
- method
- stoch_collocation
- quadrature_order

Cubature using tensor-products of Gaussian quadrature rules

Specification

Alias: none
Argument(s): INTEGERLIST

Description
Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with quadrature_order, and, optionally, dimension_preference). The default rule selection is to employ non_nested Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of p.refinement or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a dimension_preference, where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified quadrature_order and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order $p_i$ of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, $p_i$ is one less than the quadrature order $m_i$ in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, $N$, in a tensor-product expansion involving $n$ uncertain input variables is

$$N = 1 + P = \prod_{i=1}^{n}(p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the quadrature_order specification supports an array input.

sparse_grid_level

- Keywords Area
- method
- stoch_collocation
- sparse_grid_level

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): INTEGERLIST

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional(Choose One)</td>
<td>Group 1</td>
<td>restricted</td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
<tr>
<td>Optional(Choose One)</td>
<td>Group 2</td>
<td>nodal</td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hierarchical</td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
</tbody>
</table>

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with sparse_grid_level and, optionally, dimension_preference). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non_nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the dimension_preference specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full sparse_grid_level and all other dimension resolutions will be reduced in
6.2. METHOD

proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for quadrature order, the sparse_grid_level specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.

restricted

- Keywords Area
- method
- stoch_collocation
- sparse_grid_level
- restricted

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with sparse_grid_level and, optionally, dimension_preference). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non_nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the dimension_preference specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full sparse_grid_level and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for quadrature order, the sparse_grid_level specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.
unrestricted

- Keywords Area
- method
- stoch_collocation
- sparse_grid_level
- unrestricted

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where $l$ is the grid level and $m$ is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level $l$ for a sparse grid ($i = 4l + 1$) or an order $m$ for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

nodal

- Keywords Area
- method
- stoch_collocation
- sparse_grid_level
- nodal

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** nodal
6.2. METHOD

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with sparse_grid_level and, optionally, dimension_preference). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non_nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the dimension_preference specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full sparse_grid_level and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for quadrature_order, the sparse_grid_level specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.

hierarchical

- Keywords Area
- method
- stoch_collocation
- sparse_grid_level
- hierarchical

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation

Specification

Alias: none
Argument(s): none

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with sparse_grid_level and, optionally, dimension_preference). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non_nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the dimension_preference specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full sparse_grid_level and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of
tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for quadrature order, the sparse_grid_level specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.

dimension_preference

- Keywords Area
- method
- stoch_collocation
- dimension_preference

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- sobol
- decay

use_derivatives

- Keywords Area
- method
- stoch_collocation
- use_derivatives

Use derivative data to construct surrogate models
6.2. METHOD

**Specification**

**Alias:** none

- **Argument(s):** none
- **Default:** use function values only

**Description**

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

**nested**

- Keywords Area
- method
- stoch_collocation
- nested

Enforce use of nested quadrature rules if available

**Specification**

**Alias:** none

- **Argument(s):** none
- **Default:** quadrature: non_nested unless automated refinement; sparse grids: nested

**Description**

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

**non_nested**

- Keywords Area
- method
- stoch_collocation
- non_nested

Enforce use of non-nested quadrature rules

**Specification**

**Alias:** none

- **Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

**variance_based_decomp**

- Keywords Area
- method
- stoch_collocation
- variance_based_decomp

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>interaction_order</td>
<td>Specify the maximum number of variables allowed in an interaction when reporting interaction metrics. Suppresses output of sensitivity indices with values lower than this tolerance</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>drop_tolerance</td>
<td></td>
</tr>
</tbody>
</table>

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

**Default Behavior**

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol’ indices.

**Expected Outputs**

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of univariate terms for each individual variable, interaction effects represent the set of mixed terms (the complement of the univariate set), and total effects
represent the complete set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

**Usage Tips**

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

**Examples**

```plaintext
method,
    polynomial_choas # or stoch_collocation
    sparse_grid_level = 3
    variance_based_decomp interaction_order = 2
```

**Theory**

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [73]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, $S_i$, means that the uncertainty in the input variable $i$ has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [87].

**interaction_order**

- **Keywords Area**
  - method
  - stoch_collocation
  - variance_based_decomp
  - interaction_order

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** Unrestricted (VBD includes all interaction orders present in the expansion)

**Description**

The interaction_order option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous univariate_effects specification is equivalent to interaction_order = 1 in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be
diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

**drop_tolerance**
- Keywords Area
- method
- stoch_collocation
- variance_based_decomp
- drop_tolerance

Suppresses output of sensitivity indices with values lower than this tolerance

**Specification**

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

**Description**

The *drop_tolerance* keyword allows the user to specify a value below which sensitivity indices generated by *variance_based_decomp* are not displayed.

**Default Behavior**

By default, all sensitivity indices generated by *variance_based_decomp* are displayed.

**Usage Tips**

For *polynomial_choas*, which outputs main, interaction, and total effects by default, the univariate-effects may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

**Examples**

```plaintext
method,
sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

diagonal_covariance
- Keywords Area
- method
- stoch_collocation
- diagonal_covariance

Display only the diagonal terms of the covariance matrix
6.2. METHOD

**Specification**

Alias: none

- **Argument(s):** none
- **Default:** diagonal\_covariance for response vector > 10; else full\_covariance

**Description**

With a large number of responses, the covariance matrix can be very large. \texttt{diagonal\_covariance} is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

- **full\_covariance**
  - **Keywords Area**
  - **method**
  - **stoch\_collocation**
  - **full\_covariance**

  Display the full covariance matrix

**Specification**

Alias: none

- **Argument(s):** none

**Description**

With a large number of responses, the covariance matrix can be very large. \texttt{full\_covariance} is used to force Dakota to output the full covariance matrix.

- **sample\_type**
  - **Keywords Area**
  - **method**
  - **stoch\_collocation**
  - **sample\_type**

  Selection of sampling strategy

**Specification**

Alias: none

- **Argument(s):** none
- **Default:** lls
## Description

The `sample_type` keyword allows the user to select between multiple random sampling approaches. There are two primary types of sampling: Monte Carlo (pure random) and Latin Hypercube Sampling. Additionally, these methods have incremental variants that allow an existing study to be augmented with additional samples to get better estimates of mean, variance, and percentiles.

**Default Behavior**

If the `sample_type` keyword is present, it must be accompanied by `lhs`, `random`, `incremental_lhs`, or `incremental_random`. Otherwise, `lhs` will be used by default.

## Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 20
```

### lhs

- **Keywords Area**
- **method**
- **stoch_collocation**
- **sample_type**
- **lhs**

Uses Latin Hypercube Sampling (LHS) to sample variables

## Specification

**Alias:** none

**Argument(s):** none
6.2. **METHOD**

**Description**

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

**Default Behavior**

By default, Latin Hypercube Sampling is used. To explicitly specify this in the Dakota input file, however, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

**Usage Tips**

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

**Examples**

```plaintext
method sampling
   sample_type lhs
   samples = 20
```

**random**

- Keywords Area
- method
- stoch_collocation
- sample_type
- random

Uses purely random Monte Carlo sampling to sample variables

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

**Default Behavior**

Monte Carlo sampling is not used by default. To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

**Usage Tips**

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.
Examples

```
method
  sampling
    sample_type random
    samples = 200
```

**probability_refinement**

- **Keywords Area**
- **method**
- **stoch_collocation**
- **probability_refinement**

Allow refinement of probability and generalized reliability results using importance sampling.

**Topics**

This keyword is related to the topics:

- **reliability_methods**

**Specification**

**Alias:** sample_refinement  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>import</td>
<td>Sampling option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adapt_import</td>
<td>Importance sampling option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mm_adapt_import</td>
<td>Sampling option</td>
</tr>
<tr>
<td>Optional</td>
<td>refinement_samples</td>
<td></td>
<td>Specify the number of samples used to improve a probability estimate.</td>
</tr>
</tbody>
</table>

**Description**

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.
6.2. METHOD

import

- Keywords Area
- method
- stoch_collocation
- probability_refinement
- import

Sampling option

Specification

Alias: none
Argument(s): none

Description

import centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

adapt_import

- Keywords Area
- method
- stoch_collocation
- probability_refinement
- adapt_import

Importance sampling option

Specification

Alias: none
Argument(s): none

Description

adapt_import centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.
**mm_adapt_import**

- Keywords Area
- method
- stoch_collocation
- probability_refinement
- mm_adapt_import

Sampling option

**Specification**

Alias: none
Argument(s): none

**Description**

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

**refinement_samples**

- Keywords Area
- method
- stoch_collocation
- probability_refinement
- refinement_samples

Specify the number of samples used to improve a probability estimate.

**Specification**

Alias: none
Argument(s): INTEGER

**Description**

Specify the number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.
import_approx_points_file

- **Keywords Area**
- **method**
- **stoch_collocation**
- **import_approx_points_file**

Filename for points at which to evaluate the PCE/SC surrogate

**Specification**

**Alias:** none  
**Argument(s):** STRING  
**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td></td>
<td>custom_annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

**Default Behavior**  No import of points at which to evaluate the surrogate.  
**Expected Output**  The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

**Examples**

```plaintext
method polynomial_chaos
  expansion_order = 4
  import_approx_points_file = 'import.mcmc_annot.dat'
  annotated
```

**annotated**

- **Keywords Area**
- **method**
• stoch_collocation
• import_approx_points_file
• annotated

Selects annotated tabular file format

Topics
This keyword is related to the topics:
• file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips
• To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id
• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
  tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface x1   x2   obj_fn nln_ineq_con_1 nln_ineq_con_2
1   NO_ID   0.9   1.1   0.0002   0.26   0.76
2   NO_ID   0.90009 1.1 0.0001996404857  0.2601620081  0.759955
3   NO_ID   0.89991 1.1 0.0002003604863  0.2598380081  0.760045
...
```
custom.annotated

- **Keywords Area**
- method
- stoch.collocation
- import.approx.points.file
- custom.annotated

Selects custom-annotated tabular file format

### Topics

This keyword is related to the topics:

- fileformats

### Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>header</td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td>eval_id</td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td>interface_id</td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom.annotated, followed by options, in the relevant export/import context.

**Usage Tips**
• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface-id), and data for variables and responses. Input file fragment:

```
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

```
header
• Keywords Area

• method

• stoch_collocation

• import_approx_points_file

• custom_annotated

• header

Enable header row in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no header

Description
See description of parent custom_annotated
6.2. METHOD

**eval_id**

- Keywords Area
- method
- stoch_collocation
- import_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none  
  **Argument(s):** none  
  **Default:** no eval_id column

**Description**

See description of parent custom_annotated

**interface_id**

- Keywords Area
- method
- stoch_collocation
- import_approx_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none  
  **Argument(s):** none  
  **Default:** no interface_id column

**Description**

See description of parent custom_annotated
freeform

- Keywords Area
- method
- stoch_collocation
- import_approx_points_file
- freeform

Selects freeform file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
6.2. METHOD

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```python
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
  0.9   1.1   0.0002   0.26   0.76
  0.90009   1.1  0.0001996404857  0.2601620081  0.759955
  0.89991   1.1  0.0002003604863  0.2598380081  0.760045
...
```

`active_only`

- Keywords Area
- method
- stoch_collocation
- import_approx_points_file
- active_only

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.
export_approx_points_file

- Keywords Area
- method
- stoch_collocation
- export_approx_points_file

Output file for evaluations of a surrogate model

**Specification**

**Alias:** export_points_file  
**Argument(s):** STRING  
**Default:** no point export to a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional(Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

**Description**

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.

**Usage Tips**

Dakota exports tabular data in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**annotated**

- Keywords Area
- method
- stoch_collocation
- export_approx_points_file
6.2. METHOD

- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface id, specify custom.annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated

Resulting tabular file:

%eval_id interface    x1     x2     obj_fn nln_ineq_con_1 nln_ineq_con_2
1   NO_ID   0.9   1.1   0.0002   0.26   0.76
2   NO_ID   0.90009   1.1   0.0001996404857   0.2601620081   0.759955
3   NO_ID   0.89991   1.1   0.0002003604863   0.2598380081   0.760045
...
custom_annotated

- Keywords Area
- method
- stoch_collocation
- export_approx_points_file
- custom_annotated

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format

<table>
<thead>
<tr>
<th>Required-/</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>header</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header_row`, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

**Usage Tips**
Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to **annotated** format, though **freeform** remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the **freeform** option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1  x2  obj_fn nln_ineq_con_1 nln_ineq_con_2
1  0.9  1.1  0.0002  0.26  0.76
2  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...%```

```
header
```

- **Keywords Area**
- **method**
- **stoch_collocation**
- **export_approx_points_file**
- **custom_annotated**
- **header**

Enable header row in custom-annotated tabular file

### Specification

**Alias:** none

**Argument(s):** none

**Default:** no header

### Description

See description of parent **custom_annotated**
eval_id

- Keywords Area
- method
- stoch_collocation
- export_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

Alias: none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent custom_annotated

interface_id

- Keywords Area
- method
- stoch_collocation
- export_approx_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent custom_annotated
freeform

- Keywords Area
- method
- stoch_collocation
- export_approx_points_file
- freeform

Selects freeform file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
Argument(s): none
Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9     1.1   0.0002  0.26    0.76
0.90009 1.1  0.0001996404857 0.2601620081 0.759955
0.89991 1.1  0.0002003604863 0.2598380081 0.760045
...
```

### fixed_seed

- **Keywords Area**
- **method**
- **stoch_collocation**
- **fixed_seed**

Reuses the same seed value for multiple random sampling sets

### Specification

**Alias:** none

**Argument(s):** none

**Default:** not fixed; pattern varies run-to-run

### Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the coarse of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

**Default Behavior**

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

### Examples

```plaintext
method
sampling
  sample_type lhs
  samples = 10
  fixed_seed
```
6.2. METHOD

max_iterations

• Keywords Area
• method
• stoch_collocation
• max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:
• method_independent.controls

Specification
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

• Keywords Area
• method
• stoch_collocation
• convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:
• method_independent_controls

Specification
Alias: none
Argument(s): REAL
Default: 1.e-4
CHAPTER 6. KEYWORDS AREA

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

reliability_levels

- Keywords Area
- method
- stoch_collocation
- reliability_levels

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_reliability_levels</td>
<td></td>
<td>Specify which reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>
6.2. **METHOD**

**Description**
Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

**Theory**
Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_reliability_levels**
- **Keywords** Area
  - **method**
  - **stoch_collocation**
  - **reliability_levels**
  - **num_reliability_levels**

Specify which reliability_levels correspond to which response

**Specification**
**Alias:** none
**Argument(s):** INTEGRALIST
**Default:** reliability_levels evenly distributed among response functions

**Description**
See parent page

**response_levels**
- **Keywords** Area
  - **method**
  - **stoch_collocation**
  - **response_levels**

Values at which to estimate desired statistics for each response
Specification

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF probabilities/reliabilities to compute
### Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

#### Default Behavior

- If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

#### Expected Outputs

- If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

#### Usage Tips

- The `num.response_levels` is used to specify which arguments of the `response_level` correspond to which response.

### Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```plaintext
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```plaintext
method, sampling,
    samples = 100 seed = 1
```
CHAPTER 6. KEYWORDS AREA

complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
6.0e+04 6.5e+04 7.0e+04
3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std deviations = 12.4, 29.7
    descriptors = ‘TF1n’ ‘TF2n’
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = ‘TF1u’ ‘TF2u’
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = ‘TF1w’ ‘TF2w’
  histogram_bin_uncertain = 2
    num_pairs = 3 4
    abscissas = 5 8 10 .1 .2 .3 .4
    counts = 17 21 0 12 24 12 0
    descriptors = ‘TF1h’ ‘TF2h’
  histogram_point_uncertain
    real = 1
    num_pairs = 2
    abscissas = 3 4
    counts = 1 1
    descriptors = ‘TF3h’

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = ‘text_book’

responses,
  response_functions = 3
  no_gradients
  no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.760749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.419614379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.0341896485e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
</tbody>
</table>
### Level mappings for each response function:

**Complementary Cumulative Distribution Function (CCDF) for response_fn_1:**

<table>
<thead>
<tr>
<th>Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.60e+11</td>
<td>5.50e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.00e+11</td>
<td>3.80e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.40e+11</td>
<td>2.30e-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complementary Cumulative Distribution Function (CCDF) for response_fn_2:**

<table>
<thead>
<tr>
<th>Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.00e+04</td>
<td>6.10e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.50e+04</td>
<td>2.90e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.00e+04</td>
<td>9.00e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complementary Cumulative Distribution Function (CCDF) for response_fn_3:**

<table>
<thead>
<tr>
<th>Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.50e+05</td>
<td>5.20e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.00e+05</td>
<td>9.00e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.50e+05</td>
<td>0.00e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

#### num_response_levels

- Keywords Area
- method
- stoch_collocation
- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

### Specification

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** response_levels evenly distributed among response functions
CHAPTER 6. KEYWORDS AREA

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the response levels will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5

compute

  • Keywords Area
  • method
  • stoch_collocation
  • response_levels
  • compute

Selection of statistics to compute at each response level
```

Specification

Alias: none

Argument(s): none

Default: probabilities
### Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level. The `CDF/CCDF` probabilities are calculated for specified response levels using a simple binning approach. CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

#### Default Behavior

- If `response_levels` is not specified, no statistics are computed.
- If `response_levels` is specified but `compute` is not, probabilities will be computed by default.
- If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

#### Expected Output

The type of statistics specified by `compute` will be reported for each response level.

#### Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

### Examples

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
      6.0e+04 6.5e+04 7.0e+04
      3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
probabilities
```
CHAPTER 6. KEYWORDS AREA

- Keywords Area
- method
- stoch_collocation
- response_levels
- compute
- probabilities

Computes probabilities associated with response levels

**Specification**

Alias: none

**Argument(s):** none

**Description**

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

**Examples**

```plaintext
method sampling
  sample_type random
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
  6.0e+04 6.5e+04 7.0e+04
  3.5e+05 4.0e+05 4.5e+05
  compute probabilities
```

**reliabilities**

- Keywords Area
- method
- stoch_collocation
- response_levels
- compute
6.2. METHOD

- **reliabilities**

  Computes reliabilities associated with response levels

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

**Examples**

```plaintext
method sampling
  sample_type random
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
  compute reliabilities

gen_reliabilities

- **Keywords Area**

- **method**

- **stoch_collocation**

- **response_levels**

- **compute**

- **gen_reliabilities**

  Computes generalized reliabilities associated with response levels

**Specification**

**Alias:** none

**Argument(s):** none
Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior
If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs
The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
sampling
    sample_type random
    samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
compute gen_reliabilities
```

system
- **Keywords Area**
- **method**
- **stoch_collocation**
- **response_levels**
- **compute**
- **system**

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required</th>
<th>(Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/Optional Required</td>
<td>Choose One</td>
<td>Aggregate response statistics assuming a series system</td>
<td>series</td>
<td>system</td>
</tr>
</tbody>
</table>
6.2. METHOD

Description

With the system probability/reliability option, statistics for specified response levels are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

series

- Keywords Area
- method
- stoch_collocation
- response_levels
- compute
- system
- series

Aggregate response statistics assuming a series system

Specification

Alias: none
Argument(s): none

Description

See parent keyword system for description.

parallel

- Keywords Area
- method
- stoch_collocation
- response_levels
- compute
- system
• **parallel**

Aggregate response statistics assuming a parallel system

**Specification**

Alias: none  
Argument(s): none

**Description**

See parent keyword *system* for description.

distribution

• **Keywords Area**

• **method**

• **stoch_collocation**

• **distribution**

Selection of cumulative or complementary cumulative functions

**Specification**

Alias: none  
Argument(s): none  
Default: cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/-Optional Required <em>(Choose One)</em></th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>
Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

**Default Behavior**

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

**Examples**

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

cumulative

- **Keywords Area**
- **method**
- **stoch.collocation**
- **distribution**
- **cumulative**

Computes statistics according to cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.
Examples

```
method sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```

complementary

- Keywords Area
- method
- stoch_collocation
- distribution
- complementary

Computes statistics according to complementary cumulative functions

Specification

Alias: none
Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method sampling
  sample_type lhs
  samples = 10
  distribution complementary
```

probability_levels

- Keywords Area
- method
- stoch_collocation
- probability_levels

Specify probability levels at which to estimate the corresponding response value
6.2. METHOD

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

| Required/-| Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional | | num_probability_-levels | Specify which probability_-levels correspond to which response |

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If probability_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```plaintext
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
               1. .8 .5 0.
               1. .3 .2 0.
variables,
  normal_uncertain = 2
  means = 248.89, 593.33
  std deviations = 12.4, 29.7
  descriptors = ’TF1n’ ’TF2n’
  uniform_uncertain = 2
  lower_bounds = 199.3, 474.63
  upper_bounds = 298.5, 712.
  descriptors = ’TF1u’ ’TF2u’
  weibull_uncertain = 2
  alphas = 12., 30.
  betas = 250., 590.
  descriptors = ’TF1w’ ’TF2w’
histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = ’TF1h’ ’TF2h’
```
histogram_point_uncertain
real = 1
num_pairs = 2
abscissas = 3 4
counts = 1 1
descriptors = 'TF3h'

interface,
system async evaluation_concurrency = 5
analysis_driver = 'text_book'

responses,
response_functions = 3
no_gradients
no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0639753000e+11</td>
<td>5.1454122311e-12</td>
</tr>
<tr>
<td>4.0639753000e+11</td>
<td>5.4196143799e+11</td>
<td>2.433239039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.6997214153e+05</td>
<td>5.302836523e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.810096235e+05</td>
<td>9.060055634e-06</td>
</tr>
<tr>
<td>3.810096235e+05</td>
<td>4.4111498127e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.0000000000e+00</td>
<td>5.1384774972e-12</td>
<td></td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.0000000000e-01</td>
<td>5.1454122311e-12</td>
<td></td>
</tr>
<tr>
<td>4.0639753000e+11</td>
<td>3.0000000000e-01</td>
<td>2.433239039e-12</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>1.0000000000e+00</td>
<td>1.9839945149e-05</td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.0000000000e-01</td>
<td>5.8916108390e-05</td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>3.0000000000e-01</td>
<td>2.9242071306e-05</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>1.0000000000e+00</td>
<td>5.302836523e-06</td>
<td></td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.0000000000e-01</td>
<td>9.060055634e-06</td>
<td></td>
</tr>
<tr>
<td>3.810096235e+05</td>
<td>0.0000000000e+00</td>
<td>3.3274925348e-06</td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_probability_levels**

- **Keywords Area**
- **method**
- **stoch_collocation**
- **probability_levels**
- **num_probability_levels**

Specify which probability_levels correspond to which response

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** probability_levels evenly distributed among response functions

**Description**

See parent page

**gen_reliability_levels**

- **Keywords Area**
- **method**
- **stoch_collocation**
- **gen_reliability_levels**

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute
### Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

### Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_gen_reliability_levels**

- Keywords Area
- method
- stoch_collocation
- gen_reliability_levels
- num_gen_reliability_levels

Specify which `gen_reliability_levels` correspond to which response

### Specification

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** gen_reliability_levels evenly distributed among response functions

### Description

See parent page
6.2. METHOD

rng

- Keywords Area
- method
- stoch_collocation
- rng

Selection of a random number generator

Specification

Alias: none
Argument(s): none
Default: Mersenne twister (mt19937)

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rnum2</td>
<td>Generates pseudo-random numbers using the Pecos package</td>
</tr>
</tbody>
</table>

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2

mt19937

  - Keywords Area
  - method
```
Generates random numbers using the Mersenne twister

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

**Default Behavior**

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

**Examples**

```
method sampling
  sample_type lhs
  samples = 10
  seed = 98765
  rng mt19937
```

**rnum2**

- Keywords Area
- method
- stoch_collocation
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package

**Specification**

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description
The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

**Default Behavior**
rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

**Usage Tips**
Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

**samples**
- **Keywords Area**
- **method**
- **stoch_collocation**
- **samples**

Number of samples for sampling-based methods

Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

Description
The samples keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**
By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**
To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance_based_decomp, we recommend hundreds to thousands of samples. Note that for variance_based_decomp, the number of simulations performed will be N*(dim+2).

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 20
```
CHAPTER 6. KEYWORDS AREA

seed

• Keywords Area

• method

• stoch_collocation

• seed

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

method
sampling
  sample_type lhs
  samples = 10
  seed = 15347

model_pointer

• Keywords Area

• method

• stoch_collocation

• model_pointer

Identifier for model block to be used by a method
6.2. METHOD

Topics
This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SRR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SRR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
```
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system async evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.48 sampling

• Keywords Area

• method

• sampling

Randomly samples variables according to their distributions

Topics
This keyword is related to the topics:

• uncertainty quantification

• sampling

Specification
Alias: nond_sampling
Argument(s): none
### Method

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>sample_type</td>
<td>Selection of sampling strategy</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>variance_based_decomp</td>
<td>Activates global sensitivity analysis based on decomposition of response variance into contributions from variables</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>backfill</td>
<td>Ensures that the samples of discrete variables with finite support are unique</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>principal_components</td>
<td>Activates principal components analysis of the response matrix of ( N ) samples ( \times ) ( L ) responses.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>fixed_seed</td>
<td>Reuses the same seed value for multiple random sampling sets</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
</tbody>
</table>
**CHAPTER 6. KEYWORDS AREA**

| Optional | reliability_levels | Specify reliability levels at which the response values will be estimated. |
| Optional | response_levels | Values at which to estimate desired statistics for each response. |
| Optional | distribution | Selection of cumulative or complementary cumulative functions. |
| Optional | probability_levels | Specify probability levels at which to estimate the corresponding response value. |
| Optional | gen_reliability_levels | Specify generalized reliability levels at which to estimate the corresponding response value. |
| Optional | rng | Selection of a random number generator. |
| Optional | samples | Number of samples for sampling-based methods. |
| Optional | seed | Seed of the random number generator. |
| Optional | model_pointer | Identifier for model block to be used by a method. |

**Description**

This method generates parameter values by drawing samples from the specified uncertain variable probability distributions. The computational model is executed over all generated parameter values to compute the responses for which statistics are computed. The statistics support sensitivity analysis and uncertainty quantification.

**Default Behavior**

By default, sampling methods operate on aleatory and epistemic uncertain variables. The types of variables
can be restricted or expanded (to include design or state variables) through use of the `active` keyword in the `variables` block in the Dakota input file. If continuous design and/or state variables are designated as active, the sampling algorithm will treat them as parameters with uniform probability distributions between their upper and lower bounds. Refer to `variable_support` for additional information on supported variable types, with and without correlation.

The following keywords change how the samples are selected:

- `sample_type`
- `fixed_seed`
- `rng`
- `samples`
- `seed`
- `variance_based_decomp`

**Expected Outputs**

As a default, Dakota provides correlation analyses when running LHS. Correlation tables are printed with the simple, partial, and rank correlations between inputs and outputs. These can be useful to get a quick sense of how correlated the inputs are to each other, and how correlated various outputs are to inputs. `variance_based_decomp` is used to request more sensitivity information, with additional cost.

Additional statistics can be computed from the samples using the following keywords:

- `response_levels`
- `reliability_levels`
- `probability_levels`
- `gen_reliability_levels`

`response_levels` computes statistics at the specified response value. The other three allow the specification of the statistic value, and will estimate the corresponding response value.

`distribution` is used to specify whether the statistic values are from cumulative or complementary cumulative functions.

**Usage Tips**

Sampling is a robust approach to doing sensitivity analysis and uncertainty quantification that can be applied to any problem. It requires more simulations than newer, advanced methods. Thus, an alternative may be preferable if the simulation is computationally expensive.

**Examples**

```bash
# tested on Dakota 6.0 on 140501

environment
  tabular_data
  tabular_data_file = 'Sampling_basic.dat'

method
  sampling
  sample_type lhs
  samples = 20
```
model
  single

variables
  active uncertain
  uniform_uncertain = 2
  descriptors = 'input1' 'input2'
  lower_bounds = -2.0 -2.0
  upper_bounds = 2.0 2.0
  continuous_state = 1
  descriptors = 'constant1'
  initial_state = 100

interface
  analysis_drivers 'text_book'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians

This example illustrates a basic sampling Dakota input file.

- LHS is used instead of purely random sampling.
- The default random number generator is used.
- Without a seed specified, this will not be reproducible
- In the variables block, two types of variables are used
- Only the uncertain variables are varied, this is the default behavior, and is also specified by the active keyword, w/ the uncertain option

See Also
These keywords may also be of interest:

- active
- incremental_lhs

FAQ
Q: Do I need to keep the LHS* and S4 files? A: No

sample_type
  - Keywords Area
  - method
  - sampling
  - sample_type

Selection of sampling strategy
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: lhs

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>random</td>
<td>Uses purely random Monte Carlo sampling to sample variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lhs</td>
<td>Uses Latin Hypercube Sampling (LHS) to sample variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incremental_lhs</td>
<td>Augments an existing Latin Hypercube Sampling (LHS) study</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incremental_random</td>
<td>Augments an existing random sampling study</td>
</tr>
</tbody>
</table>

Description

The sample_type keyword allows the user to select between multiple random sampling approaches. There are two primary types of sampling: Monte Carlo (pure random) and Latin Hypercube Sampling. Additionally, these methods have incremental variants that allow an existing study to be augmented with additional samples to get better estimates of mean, variance, and percentiles.

Default Behavior

If the sample_type keyword is present, it must be accompanied by lhs, random, incremental_lhs, or incremental_random. Otherwise, lhs will be used by default.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 20
```

random

- Keywords Area
- method
- sampling
CHAPTER 6. KEYWORDS AREA

- sample_type
- random

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none
Argument(s): none

Description

The random keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

Monte Carlo sampling is not used by default. To change this behavior, the random keyword must be specified in conjunction with the sample_type keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

lhs

- Keywords Area
- method
- sampling
- sample_type
- lhs

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none
Argument(s): none
6.2. METHOD

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

**Default Behavior**

By default, Latin Hypercube Sampling is used. To explicitly specify this in the Dakota input file, however, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

**Usage Tips**

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 20
```

**incremental_lhs**

- Keywords Area
- method
- sampling
- sample_type
- `incremental_lhs`

Augments an existing Latin Hypercube Sampling (LHS) study

**Specification**

**Alias:** none
**Argument(s):** none
**Default:** no sample reuse in coefficient estimation

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td><code>previous_samples</code></td>
<td>Number of samples from an already-completed sampling study</td>
</tr>
</tbody>
</table>

**Description**

`incremental_lhs` will augment an existing LHS sampling study with more samples to get better estimates of mean, variance, and percentiles. The number of samples in the second set MUST currently be 2 times the number of previous samples, although incremental sampling based on any power of two may be supported in future releases.
Default Behavior

Incremental Latin Hypercube Sampling is not used by default. To change this behavior, the `incremental-lhs` keyword must be specified in conjunction with the `sample_type` keyword. Additionally, a previous LHS (or incremental LHS) sampling study with sample size $N$ must have already been performed, and the Dakota restart file must be available from this previous study. The variables and responses specifications must be the same in both studies. Incremental LHS sampling support both continuous uncertain variables and discrete uncertain variables such as discrete distributions (e.g. binomial, Poisson, etc.) as well as histogram variables and uncertain set types.

Usage Tips

The incremental approach is useful if it is uncertain how many simulations can be completed within available time.

See the examples below and the Usage and Restarting Dakota Studies pages.

Examples

For example, say a user performs an initial study using `lhs` as the `sample_type`, and generates 10 samples.

One way to ensure the restart file is saved is to specify a non-default name, via a command line option:

```bash
dakota -i LHS_10.in -w LHS_10.rst
```

which uses the input file:

```bash
# LHS_10.in

environment
  tabular_data
    tabular_data_file = 'lhs10.dat'

method
  sampling
    sample_type lhs
    samples = 10

model
  single

variables
  uniform_uncertain = 2
  descriptors = 'input1' 'input2'
  lower_bounds = -2.0 -2.0
  upper_bounds = 2.0 2.0

interface
  analysis_drivers 'text_book'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians
```

and the restart file is written to `LHS_10.rst`.

Then an incremental LHS study can be run with:

```bash
dakota -i LHS_20.in -r LHS_10.rst -w LHS_20.rst
```

where `LHS_20.in` is shown below, and `LHS_10.rst` is the restart file containing the results of the previous LHS study.
6.2. METHOD

# LHS_20.in

environment
tabular_data
tabular_data_file = 'lhs_incremental_20.dat'

method
sampling
    sample_type incremental_lhs
    samples = 20
    previous_samples = 10

model
    single

variables
    uniform_uncertain = 2
    descriptors = 'input1' 'input2'
    lower_bounds = -2.0 -2.0
    upper_bounds = 2.0  2.0

interface
    analysis_drivers 'text_book'
    fork

responses
    response_functions = 1
    no_gradients
    no_hessians

The user will get 10 new LHS samples which maintain both the correlation and stratification of the original LHS sample. The new samples will be combined with the original samples to generate a combined sample of size 20.

This is clearly seen by comparing the two tabular data files.

previous_samples

- Keywords Area
- method
- sampling
- sample_type
- incremental_lhs
- previous_samples

Number of samples from an already-completed sampling study

Specification

Alias: none
Argument(s): INTEGER
Default: 0 (no previous_samples)
CHAPTER 6. KEYWORDS AREA

Description

The `previous_samples` keyword allows the user to specify the number of samples in an existing sample set that is to be augmented using the `incremental_lhs` or `incremental_random` approach.

Default Behavior

If not specified, Dakota will assume that there are no existing samples. If specified, there must be a Dakota restart file available that contains the samples.

Examples

```
method
  sampling
    sample_type incremental_lhs
    samples = 20
    previous_samples = 10
```

`incremental_random`

- Keywords Area
- method
- sampling
- sample_type
- `incremental_random`

Augments an existing random sampling study

Specification

Alias: none  
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>previous_samples</td>
<td></td>
<td>Number of samples from an already-completed sampling study</td>
</tr>
</tbody>
</table>

Description

`incremental_random` will augment an existing random sampling study with more samples to get better estimates of mean, variance, and percentiles. There is no constraint on the number of samples in the second set as there is with incremental LHS. With `incremental_random`, one can have different sample sizes in the original and incremental set. This method can be used, for example, if you have 50 samples from a first study and find out that you are able to run 10 more samples.

Default Behavior

Incremental random sampling is not used by default. To change this behavior, the `incremental_random` keyword must be specified in conjunction with the `sample_type` keyword. Additionally, a previous random
sampling study with sample size N must have already been performed, and the dakota restart file must be available from this previous study. The variables and responses specifications must be the same in both studies.

**Usage Tips**

The incremental approach is useful if it is uncertain how many simulations can be completed within available time.

**Examples**

For example, say a user performs an initial study using *random* as the *sample_type*, and generates 50 samples. If the user creates a new input file where *samples* is now specified to be 60, the *sample_type* is defined to be *incremental_random*, and *previous_samples* is specified to be 50, the user will get 10 new random samples. The M new samples will be combined with the N previous samples to generate a combined sample of size M+N.

The method block would be the following:

```plaintext
method
  sampling
    sample_type incremental_random
    samples = 60
    previous_samples = 50
```

The syntax for running the second sample set is:

```
dakota -i input2.in -r dakota.rst
```

where `input2.in` is the file which specifies incremental sampling and `dakota.rst` is the restart file containing the results of the previous study.

**previous_samples**

- **Keywords Area**
- **method**
- **sampling**
- **sample_type**
- **incremental_random**
- **previous_samples**

Number of samples from an already-completed sampling study

**Specification**

Alias: none

**Argument(s):** INTEGER

Default: 0 (no previous_samples)

**Description**

The *previous_samples* keyword allows the user to specify the number of samples in an existing sample set that is to be augmented using the *incremental_lhs* or *incremental_random* approach.

**Default Behavior**

If not specified, Dakota will assume that there are no existing samples. If specified, there must be a Dakota restart file available that contains the samples.
Examples

```
  method
  sampling
    sample_type incremental_lhs
    samples = 20
    previous_samples = 10
```

**variance_based_decomp**

- **Keywords Area**
- **method**
- **sampling**
- **variance_based_decomp**

Activates global sensitivity analysis based on decomposition of response variance into contributions from variables.

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no variance-based decomposition

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>drop_tolerance</td>
<td>Suppresses output of sensitivity indices with values lower than this tolerance</td>
</tr>
</tbody>
</table>

**Description**

Dakota can calculate sensitivity indices through variance based decomposition using the keyword *variance_based_decomp*. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

**Default Behavior**

Because of the computational cost, *variance_based_decomp* is turned off as a default.

If the user specified a number of samples, N, and a number of nondeterministic variables, M, variance-based decomposition requires the evaluation of $N(M+2)$ samples. **Note that specifying this keyword will increase the number of function evaluations above the number requested with the samples keyword since replicated sets of sample values are evaluated.**

**Expected Outputs**

When *variance_based_decomp* is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response.

**Usage Tips**

To obtain sensitivity indices that are reasonably accurate, we recommend that N, the number of samples, be at least one hundred and preferably several hundred or thousands.
6.2. METHOD

Examples

method,
    sampling
    sample_type lhs
    samples = 100
    variance_based_decomp

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of[73] : "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, \( S_i \), means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in[73] and[87].

drop_tolerance

- Keywords Area
- method
- sampling
- variance_based_decomp
- drop_tolerance

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The drop_tolerance keyword allows the user to specify a value below which sensitivity indices generated by variance_based_decomp are not displayed.

Default Behavior

By default, all sensitivity indices generated by variance_based_decomp are displayed.

Usage Tips

For polynomial_chaos, which outputs main, interaction, and total effects by default, the univariate_effects may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).
Examples

```
method,
  sampling
    sample_type lhs
    samples = 100
    variance_based_decomp
    drop_tolerance = 0.001
```

backfill

- Keywords Area
- method
- sampling
- backfill

Ensures that the samples of discrete variables with finite support are unique

Specification

Alias: none

Argument(s): none

Description

Traditional LHS can generate replicate samples when applied to discrete variables. This keyword enforces uniqueness, which is determined only over the set of discrete variables with finite support. This allows one to generate LHS for a mixed set of continuous and discrete variables whilst still enforcing that the set of discrete LHS components of all the samples are unique.

Default Behavior

Uniqueness of samples over discrete variables is not enforced.

Usage Tips

Uniqueness can be useful when applying discrete LHS to simulations without noise.

Examples

```
method,
  sampling
    samples = 12
    seed = 123456
    sample_type lhs backfill
```

variables,

```
  active all
  uniform_uncertain = 1
  lower_bounds = 0.
  upper_bounds = 1.
  descriptors = 'continuous-uniform'
```

```
  discrete_uncertain_set
  integer = 1
  elements_per_variable = 4
  elements = 1 3 5
  descriptors = 'design-set-int'
```
6.2. METHOD

real = 1
initial_point = 0.50
set_values = 0.25 0.50 0.75 1.00
descriptors = 'design-set-real'

interface,
  direct analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

See Also

These keywords may also be of interest:

- lhs

principal_components

- Keywords Area
- method
- sampling
- principal_components

Activates principal components analysis of the response matrix of N samples * L responses.

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>percent_variance_explained</td>
<td></td>
<td>Specifies the number of components to retain to explain the specified percent variance.</td>
</tr>
</tbody>
</table>

Description

Dakota can calculate the principal components of the response matrix of N samples * L responses using the keyword principal_components. Principal components analysis (PCA) is a data reduction method. The Dakota implementation is under active development: the PCA capability may ultimately be specified elsewhere or used in different ways. For now, it is performed as a post-processing analysis based on a set of Latin Hypercube samples.

We now have field responses in Dakota. PCA is an initial approach in Dakota to analyze and represent the field data. Specifically, if we have a sample ensemble of field data responses, we want to identify the principal
components responsible for the spread of that data. Then, we can generate a surrogate model by representing
the overall response as weighted sum of M principal components, where the weights will be determined by GPs
which are a function of the input uncertain variables. This reduced form then can be used for sensitivity analysis,
calibration, etc.

The steps involved when one specifies principal_components in Dakota are as follows:

• Create an LHS input sample based on the uncertain variable specification and run the user-specified model
at the LHS points to compute the field responses. For notation purposes, there are d input parameters, N
samples, and the field length is L.

• Perform PCA on the covariance matrix of the data set from the previous step. This is done by first centering
the data (e.g. subtracting the mean of each column from that column) and performing a singular value
decomposition on the covariance matrix of the centered data. The eigenvectors of the covariance matrix
correspond to the principal components.

• Identify M principal components based on the percentage of variance explained. There is an optional key-
word for principal_components called percent_variance_explained, which is a threshold
that determines the number of components that are retained to explain at least that amount of variance. For
example, if the user specifies percent_variance_explained = 0.99, the number of components that
accounts for at least 99 percent of the variance in the responses will be retained. The default for this per-
centage is 0.95. In many applications, only a few principal components explain the majority of the variance,
resulting in significant data reduction.

• Use the principal components in a predictive sense, by constructing a prediction approximation. The basis
functions for this approximation are the principal components. The coefficients of the bases are obtained
by constructing GP surrogates for the factor scores of the M principal components. The GP surrogates will
be functions of the uncertain inputs. The idea is that we have just performed PCA on (for example) the
covariance matrix of 100 samples. Typically, those 100 samples will be generated by sampling over some d
uncertain input parameters denoted by u, so there should be a mapping from u to the field data, specifically to
the loading coefficients and the factor scores. Currently, the final item printed from a Principal Components
Analysis in Dakota is a set of prediction samples based on this prediction approximation or surrogate model
that relies on the principal components.

Default Behavior

principal_components is turned off as a default. It may be used with either scalar responses or field
responses, but it is intended to be used with large field responses as a data reduction method. For example,
typically we expect the number of LHS samples, N, to be less than the number of field responses, L (e.g. if there
is one field, the number of responses values is the length of that field).

Expected Outputs

When principal_components is specified, the number of significant principal components is printed
along with the predictions based on the principal components. If output debug is specified, additional infor-
mation is printed, including the original response matrix, the centered data, the principal components, and the
factor scores.

Usage Tips

This is a preliminary capability that is undergoing active development. Please contact the Dakota developers
team if you have problems with using this capability or want to suggest additional features.

Examples

method,
sampling
  sample_type lhs
6.2. METHOD

samples = 100
principal_components
percent_variance_explained = 0.98

Theory
There is an extensive statistical literature available on PCA. We recommend that the interested user peruse some of this in using the PCA capability.

percent_variance_explained
- Keywords Area
- method
- sampling
- principal_components
- percent_variance_explained

Specifies the number of components to retain to explain the specified percent variance.

Specification
Alias: none
Argument(s): REAL

Description
Dakota can calculate the principal components of the response matrix of N samples * L responses using the keyword principal_components. Principal components analysis (PCA) is a data reduction method. percent_variance_explained is a threshold that determines the number of components that are retained to explain at least that amount of variance. For example, if the user specifies percent_variance_explained = 0.99, the number of components that accounts for at least 99 percent of the variance in the responses will be retained. The default for this percentage is 0.95. In many applications, only a few principal components explain the majority of the variance, resulting in significant data reduction.

Expected Outputs
Usage Tips percent_variance_explained should be a real number between 0.0 and 1.0. Typically, it will be between 0.9 and 1.0.

Examples
method,
sampling
  sample_type lhs
  samples = 100
  principal_components
  percent_variance_explained = 0.98

Theory
There is an extensive statistical literature available on PCA. We recommend that the interested user peruse some of this in using the PCA capability.
fixed_seed

- Keywords Area
- method
- sampling
- fixed_seed

Reuses the same seed value for multiple random sampling sets

**Specification**

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

**Description**

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the coarse of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

**Default Behavior**

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

**Examples**

```xml
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

**max_iterations**

- Keywords Area
- method
- sampling
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls
6.2. **METHOD**

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also `max_function_evaluations`.

**convergence_tolerance**

- **Keywords Area**
- **method**
- **sampling**
- **convergence_tolerance**

Stopping criterion based on convergence of the objective function or statistics

**Topics**

This keyword is related to the topics:

- **method_independent_controls**

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** 1.e-4

**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.  
For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.  
Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

**Notes on each library:**

- **DOT:** must be satisfied for two consecutive iterations
• NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

• NL2SOL: See nl2sol

**reliability_levels**

• Keywords Area

• method

• sampling

• reliability_levels

Specify reliability levels at which the response values will be estimated

**Specification**

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_reliability_levels</td>
<td>Specify which reliability_levels correspond to which response</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).
6.2. METHOD

num_reliability_levels

- Keywords Area
- method
- sampling
- reliability_levels
- num_reliability_levels

Specify which reliability_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: reliability_levels evenly distributed among response functions

Description

See parent page

response_levels

- Keywords Area
- method
- sampling
- response_levels

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

| Required/- | Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional  | num_response_levels | num_response_levels | Number of values at which to estimate desired statistics for each response |
| Optional  | compute             | compute         | Selection of statistics to compute at each response level |
CHAPTER 6. KEYWORDS AREA

Description

The response_levels specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If response_levels are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If response_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The num_response_levels is used to specify which arguments of the response_level correspond to which response.

Examples

For example, specifying a response_level of 52.3 followed with compute probabilities will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the num_response_levels key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method, sampling, samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
6.0e+04 6.5e+04 7.0e+04
3.5e+05 4.0e+05 4.5e+05

variables, normal_uncertain = 2
means = 248.89, 593.33
std_deviations = 12.4, 29.7
descriptors = 'TF1n' 'TF2n'
uniform_uncertain = 2
lower_bounds = 199.3, 474.63
upper_bounds = 298.5, 712.

descriptors = 'TF1u' 'TF2u'
weibull_uncertain = 2
alphas = 12., 30.
betas = 250., 590.

descriptors = 'TF1w' 'TF2w'
```
histogram_bin_uncertain = 2
num_pairs = 3 4
abscissas = 5 8 10 .1 .2 .3 .4
counts = 17 21 0 12 24 12 0
descriptors = 'TF1h' 'TF2h'
histogram_point_uncertain
real = 1
num_pairs = 2
abscissas = 3 4
counts = 1 1
descriptors = 'TF3h'

interface,
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses,
response_functions = 3
no_gradients
no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.643154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.0341896485e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
<tr>
<td>3.5000000000e+05</td>
<td>4.0000000000e+05</td>
<td>8.6000000000e-06</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>4.5000000000e+05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
</table>
Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

**num_response_levels**

- **Keywords Area**
- **method**
- **sampling**
- **response_levels**
- **num_response_levels**

Number of values at which to estimate desired statistics for each response

**Specification**

Alias: none

**Argument(s):** INTEGERLIST

**Default:** response_levels evenly distributed among response functions

**Description**

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

**Default Behavior**

If `num_response_levels` is not specified, the response_levels will be evenly distributed among the responses.

**Expected Outputs**

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.
6.2. METHOD

Examples

```
method
  sampling
  samples = 100
  seed = 34785
  num_response_levels = 1 1 1
  response_levels = 0.5 0.5 0.5

compute

  • Keywords Area
  • method
  • sampling
  • response_levels
  • compute

Selection of statistics to compute at each response level
```

Specification

Alias: none

<table>
<thead>
<tr>
<th>Argument(s): none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default: probabilities</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>probabilities</td>
<td>Computes probabilities associated with response levels</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td>reliabilities</td>
<td>Computes reliabilities associated with response levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen_reliabilities</td>
<td>Computes generalized reliabilities associated with response levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
<td>Compute system reliability (series or parallel)</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach. CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
    6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
  compute reliabilities

probabilities
```

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior
If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**
The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

### Examples
```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
    6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
  compute probabilities

reliabilities
```

- **Keywords Area**
- **method**
- **sampling**
- **response_levels**
- **compute**
- **reliabilities**

Computes reliabilities associated with response levels

### Specification
**Alias:** none

**Argument(s):** none

### Description
The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

#### Default Behavior
If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

#### Expected Outputs
The Dakota output is a set of response level-reliability pairs according to the distribution defined.
Examples

```
method
sampling
  sample_type random
  samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
  6.0e+04 6.5e+04 7.0e+04
  3.5e+05 4.0e+05 4.5e+05
compute reliabilities

gen_reliabilities

- Keywords Area
- method
- sampling
- response_levels
- compute
- gen_reliabilities
```

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
sampling
  sample_type random
  samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
  6.0e+04 6.5e+04 7.0e+04
  3.5e+05 4.0e+05 4.5e+05
compute gen_reliabilities
```
system

- Keywords Area
- method
- sampling
- response_levels
- compute
- system

Compute system reliability (series or parallel)

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>series</td>
<td></td>
<td>parallel</td>
<td>Aggregate response statistics assuming a parallel system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

With the system probability/reliability option, statistics for specified response_levels are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.
• system
• series

Aggregate response statistics assuming a series system

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent keyword `system` for description.

**parallel**

• Keywords Area
• method
• sampling
• response_levels
• compute
• system
• parallel

Aggregate response statistics assuming a parallel system

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent keyword `system` for description.

**distribution**

• Keywords Area
• method
• sampling
• distribution

Selection of cumulative or complementary cumulative functions
6.2. METHOD

Specification

Alias: none
  Argument(s): none
  Default: cumulative (CDF)
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>

**Description**

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

**Default Behavior**

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

**cumulative**

- Keywords Area
- method
- sampling
- distribution
- cumulative

Computes statistics according to cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the cumulative keyword must appear in conjunction with the distribution keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

complementary

- Keywords Area
- method
- sampling
- distribution
- complementary

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must appear in conjunction with the distribution keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.
Examples

method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary

probability_levels

- Keywords Area
- method
- sampling
- probability_levels

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none
Argument(s): REALLIST
Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_probability_levels</td>
<td>none</td>
<td>Specify which probability_levels correspond to which response</td>
</tr>
</tbody>
</table>

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If probability_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
```
Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond to the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221499496e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221499496e+11</td>
<td>4.0634975300e+11</td>
<td>5.1454122311e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.419614379e+11</td>
<td>2.4334239039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.160381379e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.160381379e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.6997214153e+05</td>
<td>5.3028386523e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.8100966235e+05</td>
<td>9.0600036634e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.4111498127e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>
Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td></td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.6000000000e-01</td>
<td>6.6000000000e-01</td>
<td></td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>3.3000000000e-01</td>
<td>3.3000000000e-01</td>
<td></td>
</tr>
<tr>
<td>5.4196114379e+11</td>
<td>0.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.0000000000e-01</td>
<td>8.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>5.0000000000e-01</td>
<td>5.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>7.8702465755e+04</td>
<td>0.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td></td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.0000000000e-01</td>
<td>3.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>5.810096235e+05</td>
<td>2.0000000000e-01</td>
<td>2.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>4.4111498127e+05</td>
<td>0.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_probability_levels**

- Keywords Area
- method
- sampling
- probability_levels
- num_probability_levels

Specify which probability_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page
6.2. METHOD

**gen_reliability_levels**

- **Keywords Area**
- **method**
- **sampling**
- **gen_reliability_levels**

Specify generalized reliability levels at which to estimate the corresponding response value.

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_gen_reliability_levels</td>
<td></td>
<td>Specify which gen_reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_gen_reliability_levels**

- **Keywords Area**
- **method**
- **sampling**
- **gen_reliability_levels**
- **num_gen_reliability_levels**

Specify which gen_reliability_levels correspond to which response
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability.levels evenly distributed among response functions

Description

See parent page

rng

- Keywords Area
- method
- sampling
- rng

Selection of a random number generator

<table>
<thead>
<tr>
<th>Required/-Optional Required</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Choose One)</td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rnum2</td>
<td>Generates pseudo-random numbers using the Pecos package</td>
</tr>
</tbody>
</table>

Description

The rng keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the rng keyword must be accompanied by either rnum2 (pseudo-random numbers) or mt19937 (random numbers generated by the Mersenne twister). Otherwise, mt19937, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.
Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2

mt19937
  • Keywords Area
  • method
  • sampling
  • rng
  • mt19937

Generates random numbers using the Mersenne twister

Specification

Alias: none
  Argument(s): none

Description

The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended.

Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937

rnum2
  • Keywords Area
  • method
  • sampling
  • rng
• **rnum2**

  Generates pseudo-random numbers using the Pecos package

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

**Default Behavior**

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

**samples**

- **Keywords Area**
- **method**
- **sampling**
- **samples**

Number of samples for sampling-based methods

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** method-dependent

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**
6.2. METHOD

To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance-based_decomp, we recommend hundreds to thousands of samples. Note that for variance-based_decomp, the number of simulations performed will be N*(dim+2).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20

seed
```

- Keywords Area
- method
- sampling
- seed

Seed of the random number generator

Specification

**Alias:** none

**Argument(s):** INTEGER  
**Default:** system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**
If not specified, the seed is randomly generated.

**Expected Output**
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```
**model_pointer**

- **Keywords Area**
- **method**
- **sampling**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
```
6.2. METHOD

surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.49 importance_sampling

- Keywords Area
- method
- importance_sampling

Importance sampling

Topics
This keyword is related to the topics:

- uncertainty_quantification
- aleatory_uncertainty_quantification_methods
- sampling

Specification
Alias: nond_importance_sampling
Argument(s): none
### CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>adapt_import</td>
<td>Importance sampling option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mm_adapt_import</td>
<td>Sampling option</td>
</tr>
<tr>
<td>Optional</td>
<td>refinement_samples</td>
<td>response_levels</td>
<td>Specify the number of samples used to improve a probability estimate. Values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>distribution</td>
<td>probability_levels</td>
<td>Selection of cumulative or complementary cumulative functions</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>Specify probability levels at which to estimate the corresponding response value</td>
</tr>
</tbody>
</table>
The **importance_sampling** method is based on ideas in reliability modeling. An initial Latin Hypercube sampling is performed to generate an initial set of samples. These initial samples are augmented with samples from an importance density as follows:

- The variables are transformed to standard normal space.
- In the transformed space, the importance density is a set of normal densities centered around points which are in the failure region.
- Note that this is similar in spirit to the reliability methods, in which importance sampling is centered around a Most Probable Point (MPP).
- In the case of the LHS samples, the importance sampling density will simply by a mixture of normal distributions centered around points in the failure region.

**Options**

Choose one of the importance sampling options:

- `import`
- `adapt_import`
- `mm_adapt_import`

The options for importance sampling are as follows: `import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).
adapt_import is the same as import but is performed iteratively until the failure probability estimate converges. mm_adapt_import starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then mm_adapt_import proceeds similarly to adapt_import (sample until convergence).

Theory

Importance sampling is a method that allows one to estimate statistical quantities such as failure probabilities (e.g. the probability that a response quantity will exceed a threshold or fall below a threshold value) in a way that is more efficient than Monte Carlo sampling. The core idea in importance sampling is that one generates samples that preferentially samples important regions in the space (e.g. in or near the failure region or user-defined region of interest), and then appropriately weights the samples to obtain an unbiased estimate of the failure probability [76]. In importance sampling, the samples are generated from a density which is called the importance density: it is not the original probability density of the input distributions. The importance density should be centered near the failure region of interest. For black-box simulations such as those commonly interfaced with Dakota, it is difficult to specify the importance density a priori: the user often does not know where the failure region lies, especially in a high-dimensional space.[78]. We have developed two importance sampling approaches which do not rely on the user explicitly specifying an importance density.

See Also

These keywords may also be of interest:

- adaptive_sampling
- gpais
- local_reliability
- global_reliability
- sampling
- polynomial_chaos
- stoch_collocation

import

- Keywords Area
- method
- importance_sampling
- import

Sampling option
6.2. METHOD

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

import centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

**adapt_import**

- **Keywords Area**
- **method**
- **importance_sampling**
- **adapt_import**

Importance sampling option

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

adapt_import centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

**mm_adapt_import**

- **Keywords Area**
- **method**
- **importance_sampling**
- **mm_adapt_import**

Sampling option

**Specification**

**Alias:** none

**Argument(s):** none
**Description**

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

**refinement_samples**

- Keywords Area
- method
- importance_sampling
- refinement_samples

Specify the number of samples used to improve a probability estimate.

**Specification**

Alias: none

*Argument(s):* INTEGER

**Description**

Specify the number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

**response_levels**

- Keywords Area
- method
- importance_sampling
- response_levels

Values at which to estimate desired statistics for each response.

**Specification**

Alias: none

*Argument(s):* REALLIST

*Default:* No CDF/CCDF probabilities/reliabilities to compute
### Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

**Default Behavior**

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

**Expected Outputs**

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

**Usage Tips**

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

### Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```plaintext
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```plaintext
method, sampling,
samples = 100 seed = 1
```
Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

### Probability Density Function (PDF) histograms for each response function:

#### PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

#### PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.0341896485e-05</td>
</tr>
</tbody>
</table>

#### PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
</tbody>
</table>
3.5000000000e+05  4.0000000000e+05  8.6000000000e-06
4.0000000000e+05  4.5000000000e+05  1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

**num_response_levels**

- **Keywords Area**
- **method**
- **importance_sampling**
- **response_levels**
- **num_response_levels**

Number of values at which to estimate desired statistics for each response

Specification

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** response_levels evenly distributed among response functions
**Description**

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

**Default Behavior**

If `num_response_levels` is not specified, the response levels will be evenly distributed among the responses.

**Expected Outputs**

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

**Examples**

```plaintext
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5

compute
```

- **Keywords Area**
- **method**
- **importance_sampling**
- **response_levels**
- **compute**

Selection of statistics to compute at each response level

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** probabilities
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td>probabilities</td>
<td>Computes probabilities associated with response levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gen_reliabilities</td>
<td>Computes generalized reliabilities associated with response levels</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>system</td>
<td>Compute system reliability (series or parallel)</td>
</tr>
</tbody>
</table>

### Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

#### Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

#### Expected Output

The type of statistics specified by `compute` will be reported for each response level.

#### Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

### Examples

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                        6.0e+04 6.5e+04 7.0e+04
                        3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
probabilities
```

- **Keywords Area**
- **method**
- **importance_sampling**
• response_levels
• compute
• probabilities

Computes probabilities associated with response levels

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

**Examples**

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                         6.0e+04 6.5e+04 7.0e+04
                         3.5e+05 4.0e+05 4.5e+05
  compute probabilities

gen_reliabilities
```

Computes generalized reliabilities associated with response levels
6.2. METHOD

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method sampling
  sample_type random
  samples = 100 seed = 1
  complementary_distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

```
method system
  Keywords Area
  method
  importance_sampling
  response_levels
  compute
  system

Compute system reliability (series or parallel)
```

Specification

Alias: none

Argument(s): none


<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>series</td>
<td>Aggregate response statistics assuming a series system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parallel</td>
<td>Aggregate response statistics assuming a parallel system</td>
</tr>
</tbody>
</table>

**Description**

With the system probability/reliability option, statistics for specified \textit{response\_levels} are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

- \textit{series}
- \textit{Keywords Area}
- \textit{method}
- \textit{importance\_sampling}
- \textit{response\_levels}
- \textit{compute}
- \textit{system}
- \textit{series}

Aggregate response statistics assuming a series system

**Specification**

Alias: none

Argument(s): none

**Description**

See parent keyword \textit{system} for description.
6.2. METHOD

parallel

- Keywords Area
- method
- importance_sampling
- response_levels
- compute
- system
- parallel

Aggregate response statistics assuming a parallel system

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent keyword system for description.

**max_iterations**

- Keywords Area
- method
- importance_sampling
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also max_function_evaluations.
convergence_tolerance

- Keywords Area
- method
- importance_sampling
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:
- method_independent_controls

Specification

Alias: none
- Argument(s): REAL
- Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol
6.2. METHOD

distribution

- Keywords Area
- method
- importance_sampling
- distribution

Selection of cumulative or complementary cumulative functions

Specification

Alias: none
Argument(s): none
Default: cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>

Description

The distribution keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the distribution keyword is present, it must be accompanied by either cumulative or complementary. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```
cumulative

- Keywords Area
- method
- importance_sampling
- distribution
- cumulative

Computes statistics according to cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

**Examples**

```plaintext
method sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```

complementary

- Keywords Area
- method
- importance_sampling
- distribution
- complementary

Computes statistics according to complementary cumulative functions
6.2. METHOD

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must be appear in conjunction with the distribution keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary

probability_levels

- Keywords Area
- method
- importance_sampling
- probability_levels

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_probability_levels</td>
<td></td>
<td>Specify which probability_levels correspond to which response</td>
</tr>
</tbody>
</table>
**CHAPTER 6. KEYWORDS AREA**

**Description**

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

**Expected Output**

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

**Examples**

The Dakota input file below specifies a sampling method with probability levels of interest.

```plaintext
method,
  sampling,
    samples = 100 seed = 1
  complementary distribution
    probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.
variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = 'TF1h' 'TF2h'
histogram_point_uncertain
  real = 1
  num_pairs = 2
  abscissas = 3 4
  counts = 1 1
  descriptors = 'TF3h'
interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'
responses,
  response_functions = 3
  no_gradients
  no_hessians
```
6.2. METHOD

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated.

Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0634975300e+11</td>
<td>5.145122311e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.4196114379e+11</td>
<td>2.4334239039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.6997214153e+05</td>
<td>5.3028386523e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.8100966235e+05</td>
<td>9.0600056346e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.4111498227e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>3.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4196114379e+11</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>3.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.8702465755e+04</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>2.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4111498227e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_probability_levels
• Keywords Area
• method
• importance_sampling
• probability_levels
• num_probability_levels

Specify which probability_levels correspond to which response

**Specification**

Alias: none

**Argument(s):** INTEGERLIST

**Default:** probability_levels evenly distributed among response functions

**Description**

See parent page

**gen_reliability_levels**

• Keywords Area
• method
• importance_sampling
• gen_reliability_levels

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

Alias: none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute

| Required/- | Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional | | num_gen_reliability_levels | Specify which gen_reliability_levels correspond to which response |

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).
Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function. In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_gen_reliability_levels

- Keywords Area
- method
- importance_sampling
- gen_reliability_levels
- num_gen_reliability_levels

Specify which gen_reliability_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability_levels evenly distributed among response functions

Description

See parent page

rng

- Keywords Area
- method
- importance_sampling
- rng

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister ( mt19937 )
### Keywords Area

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rnum2</td>
<td>Generates pseudo-random numbers using the Pecos package</td>
</tr>
</tbody>
</table>

### Description

The `rng` keyword is used to indicate a choice of random number generator.

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**

The default is recommended, as the Mersenne twister is a higher quality random number generator.

### Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

**mt19937**

- Keywords Area
- method
- importance_sampling
- rng
- mt19937

Generates random numbers using the Mersenne twister

### Specification

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description
The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior
mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

Usage Tips
Use of the Mersenne twister random number generator (mt19937) is recommended.

Examples
method sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937

rnum2
- Keywords Area
- method
- importance_sampling
- rng
- rnum2
  Generates pseudo-random numbers using the Pecos package

Specification
Alias: none
Argument(s): none

Description
The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior
rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

Usage Tips
Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples
method sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
samples

- Keywords Area
- method
- importance_sampling
- samples

Number of samples for sampling-based methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where “dim” is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be N*(dim+2).

**Examples**

```plaintext
method
sampling
  sample_type lhs
  samples = 20
```

**seed**

- Keywords Area
- method
- importance_sampling
- seed

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)
Description
The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples
```yaml
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

model_pointer
- Keywords Area
- method
- importance_sampling
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:
- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.
**Usage Tips**
When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.
See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6
  sample_type lhs
distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0.  0.
  upper_bounds = 1.  1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

### 6.2.50 gpaia

- **Keywords Area**
6.2. METHOD

- method
- gpais

Gaussian Process Adaptive Importance Sampling

Topics

This keyword is related to the topics:

- uncertainty_quantification

Specification

Alias: gaussian_process_adaptive_importance_sampling

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>emulator_samples</td>
<td>Number of data points used to train the surrogate model or emulator</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build-points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>export_approx-points_file</td>
<td>Output file for evaluations of a surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>response_levels</td>
<td>Values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>distribution</td>
<td>Selection of cumulative or complementary cumulative functions</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| Optional | probability_levels | Specify probability levels at which to estimate the corresponding response value. |
| Optional | gen_reliability_levels | Specify generalized reliability levels at which to estimate the corresponding response value. |
| Optional | rng | Selection of a random number generator. |
| Optional | samples | Number of samples for sampling-based methods. |
| Optional | seed | Seed of the random number generator. |
| Optional | model_pointer | Identifier for model block to be used by a method. |

Description

gpais is recommended for problems that have a relatively small number of input variables (e.g. less than 10-20). This method, Gaussian Process Adaptive Importance Sampling, is outlined in the paper[?].

This method starts with an initial set of LHS samples and adds samples one at a time, with the goal of adaptively improving the estimate of the ideal importance density during the process. The approach uses a mixture of component densities. An iterative process is used to construct the sequence of improving component densities. At each iteration, a Gaussian process (GP) surrogate is used to help identify areas in the space where failure is likely to occur. The GPs are not used to directly calculate the failure probability; they are only used to approximate the importance density. Thus, the Gaussian process adaptive importance sampling algorithm overcomes limitations involving using a potentially inaccurate surrogate model directly in importance sampling calculations.

See Also

These keywords may also be of interest:

- adaptive_sampling
- local_reliability
- global_reliability
- sampling
6.2. METHOD

- importance_sampling
- polynomial_chaos
- stoch_collocation

emulator_samples

- Keywords Area
- method
- gpais
- emulator_samples

Number of data points used to train the surrogate model or emulator

Specification

Alias: none

Argument(s): INTEGER
Default: 10000

Description

This keyword refers to the number of build points or training points used to construct a Gaussian process emulator. If the user specifies a number of emulator_samples that is less than the minimum number of points required to build the GP surrogate, Dakota will augment the samples to obtain the minimum required.

import_build_points_file

- Keywords Area
- method
- gpais
- import_build_points_file

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING
Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


### Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

#### Default Behavior

By default, methods do not import points from a file.

#### Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom.annotated
- freeform

### Examples

```plaintext
method
polynomial_chaos
  expansion_order = 4
  import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
  Keywords Area
  method
  gpaix
  import_build_points_file
  annotated
  Selects annotated tabular file format
```
6.2. METHOD

Topics
This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```
custom_annotated

- Keywords Area
- method
- gpais
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

**Usage Tips**
6.2. METHOD

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1  x2  obj_fn  nln_ineq_con_1  nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

- **header**
  - **Keywords Area**
  - **method**
  - **gpais**
  - **import_build_points_file**
  - **custom_annotated**
  - **header**

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent custom_annotated


**eval_id**

- **Keywords Area**
- **method**
- **gpais**
- **import_build_points_file**
- **custom_annotated**
- **eval_id**

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent *custom_annotated*

**interface_id**

- **Keywords Area**
- **method**
- **gpais**
- **import_build_points_file**
- **custom_annotated**
- **interface_id**

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent *custom_annotated*
6.2. METHOD

freeform

- Keywords Area
- method
- gpais
- import_build_points_file
- freeform

Selects freeform file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```python
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

<table>
<thead>
<tr>
<th></th>
<th>1.1</th>
<th>0.0002</th>
<th>0.26</th>
<th>0.76</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`active_only`

- **Keywords Area**
- **method**
- **gpais**
- **import_build_points_file**
- **active_only**

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- **file_formats**

Specification

**Alias**: none

**Argument(s)**: none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.
export_approx_points_file

- **Keywords Area**
- **method**
- **gpais**
- **export_approx_points_file**

Output file for evaluations of a surrogate model

### Specification

**Alias:** export_points_file  
**Argument(s):** STRING  
**Default:** no point export to a file

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

### Description

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.

**Usage Tips**

Dakota exports tabular data in one of three formats:

- **annotated** (default)
- **custom_annotated**
- **freeform**

**annotated**

- **Keywords Area**
- **method**
- **gpais**
- **export_approx_points_file**
• **annotated**

Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

• **file formats**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format

**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The **annotated** keyword can be used to explicitly specify this.

**Usage Tips**

• To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify **custom_annotated**

  header eval_id

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to **annotated** format, though **freeform** remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the **freeform** option.

**Examples**

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface   x1    x2    obj_fn  nln_ineq_con_1 nln_ineq_con_2
1  NO_ID    0.9   1.1  0.0002    0.26   0.76
2  NO_ID  0.90009 1.1  0.0001996404857 0.2601620081 0.759955
3  NO_ID  0.89991 1.1  0.0002003604863 0.2598380081 0.760045
...```
6.2. METHOD

custom_annotated

- Keywords Area
- method
- gpais
- export_approx_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
Arguments: none
Default: annotated format

| Required/- | Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional   |                      | header         | Enable header row in custom-annotated tabular file |
| Optional   |                      | eval_id        | Enable evaluation ID column in custom-annotated tabular file |
| Optional   |                      | interface_id   | Enable interface ID column in custom-annotated tabular file |

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips
• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id  x1    x2    obj_fn  nln_ineq_con_1  nln_ineq_con_2
1         0.9    1.1    0.0002  0.26      0.76
2         0.90009 1.1    0.0001996404857 0.2601620081 0.759955
3         0.89991 1.1    0.0002003604863 0.2598380081 0.760045
...```

header

• Keywords Area

• method

• gpais

• export_approx_points_file

• custom_annotated

• header

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent custom_annotated
6.2. METHOD

eval_id

- Keywords Area
- method
- gpais
- export_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no eval_id column

Description
See description of parent custom_annotated

interface_id

- Keywords Area
- method
- gpais
- export_approx_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no interface_id column

Description
See description of parent custom_annotated
freeform

- Keywords Area

- method

- gpais

- export_approx_points_file

- freeform

Selects freeform file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
freeform

Resulting tabular file:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>1.1</td>
<td>0.002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>0.9009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**response_levels**

- **Keywords Area**
- **method**
- **gpais**
- **response_levels**

Values at which to estimate desired statistics for each response

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF probabilities/reliabilities to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_response_levels</td>
<td>num_response_levels</td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td></td>
<td>compute</td>
<td>compute</td>
<td>Selection of statistics to compute at each response level</td>
</tr>
</tbody>
</table>

**Description**

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

**Default Behavior**

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

**Expected Outputs**
If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

**Usage Tips**

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

**Examples**

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```plaintext
response_levels = 1. 2..1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```plaintext
method,
  sampling,
    samples = 100 seed = 1
    complementary distribution
      response_levels = 3.6e+11 4.0e+11 4.4e+11
      6.0e+04 6.5e+04 7.0e+04
      3.5e+05 4.0e+05 4.5e+05
variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = ’TF1n’ ’TF2n’
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = ’TF1u’ ’TF2u’
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = ’TF1w’ ’TF2w’
  histogram_bin_uncertain = 2
    num_pairs = 3 4
    abscissas = 5 8 10 .1 .2 .3 .4
    counts = 17 21 0 12 24 12 0
    descriptors = ’TF1h’ ’TF2h’
  histogram_point_uncertain
    real = 1
    num_pairs = 2
    abscissas = 3 4
    counts = 1 1
```
Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

**PDF for response_fn_1:**

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.360173194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

**PDF for response_fn_2:**

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.034189685e-05</td>
</tr>
</tbody>
</table>

**PDF for response_fn_3:**

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
<tr>
<td>3.5000000000e+05</td>
<td>4.0000000000e+05</td>
<td>8.6000000000e-06</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>4.5000000000e+05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

**Complementary Cumulative Distribution Function (CCDF) for response_fn_1:**

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complementary Cumulative Distribution Function (CCDF) for response_fn_2:**

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complementary Cumulative Distribution Function (CCDF) for response_fn_3:**

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.
In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

**num_response_levels**

- Keywords Area
- method
- gpais
- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

**Specification**

Alias: none

**Argument(s):** INTEGERLIST

**Default:** response_levels evenly distributed among response functions

**Description**

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

**Default Behavior**

If `num_response_levels` is not specified, the response_levels will be evenly distributed among the responses.

**Expected Outputs**

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

**Examples**

```
method
sampling
  samples = 100
  seed = 34785
num_response_levels = 1 1 1
response_levels = 0.5 0.5 0.5
```
compute

- Keywords Area
- method
- gpais
- response_levels
- compute

Selection of statistics to compute at each response level

**Specification**

**Alias**: none
**Argument(s)**: none
**Default**: probabilities

<table>
<thead>
<tr>
<th>Required/Optional Required/(Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>probabilities</td>
<td>Computes probabilities associated with response levels</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>gen_reliabilities</td>
<td>Computes generalized reliabilities associated with response levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
<td>Compute system reliability (series or parallel)</td>
</tr>
</tbody>
</table>

**Description**

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

**Default Behavior**

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

**Expected Output**

The type of statistics specified by `compute` will be reported for each response level.

**Usage Tips**

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach. CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.
Examples
method
sampling
  sample_type random
  samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
  6.0e+04 6.5e+05 7.0e+04
  3.5e+05 4.0e+05 4.5e+05
compute probabilities

probabilities
  Keywords Area
  method
  gpais
  response_levels
  compute
  probabilities
Computes probabilities associated with response levels

Specification
Alias: none
Argument(s): none

Description
The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior
If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs
The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples
method
sampling
  sample_type random
  samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
  6.0e+04 6.5e+05 7.0e+04
  3.5e+05 4.0e+05 4.5e+05
compute probabilities
6.2. \textit{METHOD}

\texttt{gen\_reliabilities}

- \texttt{Keywords Area}
- \texttt{method}
- \texttt{gpais}
- \texttt{response\_levels}
- \texttt{compute}
- \texttt{gen\_reliabilities}

Computes generalized reliabilities associated with response levels

\textbf{Specification}

\textbf{Alias}: none
\textbf{Argument(s)}: none

\textbf{Description}

The \texttt{gen\_reliabilities} keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

\textbf{Default Behavior}

If \texttt{response\_levels} is specified, the generalized reliabilities are not computed by default. To change this behavior, the \texttt{gen\_reliabilities} keyword should be specified in conjunction with the \texttt{compute} keyword.

\textbf{Expected Outputs}

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

\textbf{Examples}

\texttt{method sampling}
\texttt{\hspace{1cm} sample\_type random}
\texttt{\hspace{1cm} samples = 100 seed = 1}
\texttt{\hspace{1cm} complementary distribution}
\texttt{\hspace{1cm} response\_levels = \texttt{3.6e+11 4.0e+11 4.4e+11}}
\texttt{\hspace{2cm} 6.0e+04 6.5e+04 7.0e+04}}
\texttt{\hspace{3cm} 3.5e+05 4.0e+05 4.5e+05}}
\texttt{\hspace{1cm} compute \texttt{gen\_reliabilities}}

\texttt{system}

- \texttt{Keywords Area}
- \texttt{method}
- \texttt{gpais}
- \texttt{response\_levels}
- \texttt{compute}
- \texttt{system}

Compute system reliability (series or parallel)
**Specification**

Alias: none  
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>series</td>
<td>Aggregate response statistics assuming a series system</td>
</tr>
<tr>
<td>Required(Choose One)</td>
<td>Group 1</td>
<td>parallel</td>
<td>Aggregate response statistics assuming a parallel system</td>
</tr>
</tbody>
</table>

**Description**

With the system probability/reliability option, statistics for specified response levels are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

series
- Keywords Area
- method
- gpais
- response_levels
- compute
- system
- series

Aggregate response statistics assuming a series system

**Specification**

Alias: none  
Argument(s): none

**Description**

See parent keyword system for description.
6.2. METHOD

parallel

- Keywords Area
- method
- gpais
- response_levels
- compute
- system
- parallel

Aggregate response statistics assuming a parallel system

**Specification**

*Alias:* none

*Argument(s):* none

**Description**

See parent keyword *system* for description.

**max_iterations**

- Keywords Area
- method
- gpais
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

*Alias:* none

*Argument(s):* INTEGER

*Default:* 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also *max_function_evaluations*. 
distribution

- Keywords Area
- method
- gpais
- distribution

Selection of cumulative or complementary cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/Optional Required</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose One</td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>

**Description**

The *distribution* keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

**Default Behavior**

If the *distribution* keyword is present, it must be accompanied by either *cumulative* or *complementary*. Otherwise, a cumulative distribution will be used by default.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

**Examples**

```
method
  sampling
    sample_type lhs
    samples = 10
  distribution cumulative
```
6.2. METHOD

cumulative

- Keywords Area
- method
- gpais
- distribution
- cumulative

Computes statistics according to cumulative functions

Specification

Alias: none
Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the cumulative keyword must appear in conjunction with the distribution keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

method
sampling
sample_type lhs
samples = 10
distribution cumulative

complementary

- Keywords Area
- method
- gpais
- distribution
- complementary

Computes statistics according to complementary cumulative functions
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior
By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must be appear in conjunction with the distribution keyword.

Expected Outputs
Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

`probability_levels`

- **Keywords Area**
- **method**
- **gpais**
- **probability_levels**

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none
Argument(s): REALLIST
Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>num_probability-</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>levels</td>
<td>Specify which</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>probability_levels</th>
<th>correspond to which response</th>
</tr>
</thead>
</table>
6.2. METHOD

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If probability_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method, sampling,
samples = 100 seed = 1
complementary distribution
probability_levels = 1. .66 .33 0.
1. .8 .9 0.
1. .3 .2 0.
variables,
normal_uncertain = 2
means = 248.89, 593.33
std_deviations = 12.4, 29.7
descriptors = ´TF1n´ ´TF2n´
uniform_uncertain = 2
lower_bounds = 199.3, 474.63
upper_bounds = 298.5, 712.
descriptors = ´TF1u´ ´TF2u´
weibull_uncertain = 2
alphas = 12., 30.
betas = 250., 590.
descriptors = ´TF1w´ ´TF2w´
histogram_bin_uncertain = 2
num_pairs = 3 4
abscissas = 5 8 10 .1 .2 .3 .4
counts = 17 21 0 12 24 12 0
descriptors = ´TF1h´ ´TF2h´
histogram_point_uncertain
real = 1
num_pairs = 2
abscissas = 3 4
counts = 1 1
descriptors = ´TF3h´
interface,
system async evaluation_concurrency = 5
analysis_driver = ´text_book´
responses,
response_functions = 3
no_gradients
no_hessians
```
Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:
Bin Lower     Bin Upper     Density Value
----------     ----------     ------------
2.7604749078e+11  3.4221494996e+11  5.1384774972e-12
3.4221494996e+11  4.0634975300e+11  5.145122311e-12
4.0634975300e+11  5.4196114379e+11  2.4334239039e-12
PDF for response_fn_2:
Bin Lower     Bin Upper     Density Value
----------     ----------     ------------
4.6431154744e+04  5.6511827775e+04  1.9839945149e-05
5.6511827775e+04  6.1603813790e+04  5.8916108390e-05
6.1603813790e+04  7.8702465755e+04  2.9242071306e-05
PDF for response_fn_3:
Bin Lower     Bin Upper     Density Value
----------     ----------     ------------
2.379637090e+05  3.6997214153e+05  5.3028386523e-06
3.6997214153e+05  3.8100966235e+05  9.060055634e-06
3.8100966235e+05  4.411498127e+05  3.3274925348e-06
Level mappings for each response function:
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
Response Level     Probability Level     Reliability Index     General Rel Index
--------------     -----------------     -----------------     ----------------
2.7604749078e+11  1.0000000000e+00
3.4221494996e+11  6.6000000000e-01
4.0634975300e+11  3.3000000000e-01
5.4196114379e+11  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
Response Level     Probability Level     Reliability Index     General Rel Index
--------------     -----------------     -----------------     ----------------
4.6431154744e+04  1.0000000000e+00
5.6511827775e+04  8.0000000000e-01
6.1603813790e+04  5.0000000000e-01
7.8702465755e+04  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
Response Level     Probability Level     Reliability Index     General Rel Index
--------------     -----------------     -----------------     ----------------
2.379637090e+05  1.0000000000e+00
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.411498127e+05  0.0000000000e+00

Theory
Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_probability_levels
### 6.2. METHOD

- Keywords Area
- method
- gpais
- probability_levels
- num_probability_levels

Specify which `probability_levels` correspond to which response

**Specification**

**Alias:** none  
**Argument(s):** INTEGERLIST  
**Default:** probability_levels evenly distributed among response functions

**Description**

See parent page

**gen_reliability_levels**

- Keywords Area
- method
- gpais
- `gen_reliability_levels`

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
</table>
| Optional           |                      | `num_gen_reliability_levels`  
|                    |                      | Specify which `gen_reliability_levels` correspond to which response

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).
Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function. In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_gen_reliability_levels**

- Keywords Area
- method
- gpais
- gen_reliability_levels
- num_gen_reliability_levels

Specify which gen_reliability_levels correspond to which response

**Specification**

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability_levels evenly distributed among response functions

**Description**

See parent page

**rng**

- Keywords Area
- method
- gpais
- rng

Selection of a random number generator

**Specification**

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
</tbody>
</table>

### Description

The `rng` keyword is used to indicate a choice of random number generator.

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**

The default is recommended, as the Mersenne twister is a higher quality random number generator.

### Examples

```dakota
method sampling
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
  rng rnum2
```

### mt19937

- **Keywords Area**

### Specification

**Alias:** none

**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description

The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on Wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

**Default Behavior**

mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (mt19937) is recommended.

Examples

```
method sampling
   sample_type lhs
   samples = 10
   seed = 98765
   rng mt19937
```

rnum2

- Keywords Area
- method
- gpais
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package

Specification

**Alias:** none

**Argument(s):** none

Description

The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

**Default Behavior**

rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples

```
method sampling
   sample_type lhs
   samples = 10
   seed = 98765
   rng rnum2
```
6.2. METHOD

samples

- Keywords Area
- method
- gpais
- samples

Number of samples for sampling-based methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

**Description**

The *samples* keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least *dim+1* samples should be used, where "*dim*" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least *(dim+1)(dim+2)/2* samples are needed. For uncertainty quantification, we recommend at least 10*dim* samples. For *variance_based_decomp*, we recommend hundreds to thousands of samples. Note that for *variance_based_decomp*, the number of simulations performed will be N*(dim+2).

**Examples**

```
method sampling
   sample_type lhs
   samples = 20
```

**seed**

- Keywords Area
- method
- gpais
- seed

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)
CHAPTER 6. KEYWORDS AREA

Description
The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

model_pointer
  • Keywords Area
  • method
  • gpais
  • model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:
  • block_pointer

Specification
Alias: none
  Argument(s): STRING
  Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.
6.2. METHOD

**Usage Tips**
When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.
See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'Surr'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'Surr'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.51 adaptive_sampling

- Keywords Area
• method
• adaptive_sampling

(Experimental) Build a GP surrogate and refine it adaptively

Topics
This keyword is related to the topics:
• uncertainty_quantification

Specification
Alias: nond_adaptive_sampling
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>emulator_samples</td>
<td>Number of data points used to train the surrogate model or emulator</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>fitness_metric</td>
<td>(Experimental) Specify the fitness_metric used to select the next point</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>batch_selection</td>
<td>How to select new points (Experimental) How to select new points</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>batch_size</td>
<td>The number of points to add in each batch.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>export_approx_points_file</td>
<td>Output file for evaluations of a surrogate model</td>
</tr>
</tbody>
</table>
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>response_levels</th>
<th>Values at which to estimate desired statistics for each response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>misc_options</td>
<td>(Experimental) This is a capability used to send the adaptive sampling algorithm specific options.</td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>distribution</td>
<td>Selection of cumulative or complementary cumulative functions</td>
</tr>
<tr>
<td>Optional</td>
<td>probability_levels</td>
<td>Specify probability levels at which to estimate the corresponding response value</td>
</tr>
<tr>
<td>Optional</td>
<td>gen_reliability_levels</td>
<td>Specify generalized reliability levels at which to estimate the corresponding response value</td>
</tr>
<tr>
<td>Optional</td>
<td>rng</td>
<td>Selection of a random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td>samples</td>
<td>Number of samples for sampling-based methods</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>seed</th>
<th>Seed of the random number generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

### Description

This is an experimental capability that is not ready for production use at this point.

The goal in performing adaptive sampling is to construct a surrogate model that can be used as an accurate predictor to some expensive simulation, thus it is to one’s advantage to build a surrogate that minimizes the error over the entire domain of interest using as little data as possible from the expensive simulation. The adaptive part alludes to the fact that the surrogate will be refined by focusing samples of the expensive simulation on particular areas of interest rather than rely on random selection or standard space-filling techniques.

At a high-level, the adaptive sampling pipeline is a four-step process:

- Evaluate the expensive simulation (referred to as the true model) at initial sample point
  1. Fit a surrogate model
  2. Create a candidate set and score based on information from surrogate
  3. Select a candidate point to evaluate the true model
  4. Loop until done

In terms of the Dakota implementation, the adaptive sampling method currently uses Latin Hypercube sampling (LHS) to generate the initial points in Step 1 above. For Step 2, we use a Gaussian process model.

The default behavior is to add one point at a time. At each iteration (e.g. each loop of Steps 2-4 above), a Latin Hypercube sample is generated (a new one, different from the initial sample) and the surrogate model is evaluated at this points. These are the candidate points that are then evaluated according to the fitness metric. The number of candidates used in practice should be high enough to fill most of the input domain: we recommend at least hundreds of points for a low-dimensional problem. All of the candidates (samples on the emulator) are given a score and then the highest-scoring candidate is selected to be evaluated on the true model.

The adaptive sampling method also can generate batches of points to add at a time using the `batch_-selection` and `batch_size` keywords.

### See Also

These keywords may also be of interest:

- `gpais`
- `local_reliability`
- `global_reliability`
- `sampling`
- `importance_sampling`
- `polynomial_choas`
- `stoch_collocation`
6.2. METHOD

emulator_samples

- Keywords Area
- method
- adaptive_sampling
- emulator_samples

Number of data points used to train the surrogate model or emulator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 400

**Description**

This keyword refers to the number of build points or training points used to construct a Gaussian process emulator. If the user specifies a number of emulator_samples that is less than the minimum number of points required to build the GP surrogate, Dakota will augment the samples to obtain the minimum required.

fitness_metric

- Keywords Area
- method
- adaptive_sampling
- fitness_metric

(Experimental) Specify the fitness_metric used to select the next point

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** predicted_variance

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required(Choose One)</td>
<td>Group 1</td>
<td>predicted_variance</td>
<td>Pick points with highest variance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>distance</td>
<td>Space filling metric</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gradient</td>
<td>Fill the range space of the surrogate</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

The adaptive sampling is an experimental capability that is not ready for production use at this time. The user can specify the fitness_metric used to select the next point (or points) to evaluate and add to the set. The fitness metrics used for scoring candidate points include:

- predicted_variance
- distance
- gradient

predicted_variance

- Keywords Area
- method
- adaptive_sampling
- fitness_metric
- predicted_variance

Pick points with highest variance

Specification

Alias: none
Argument(s): none

Description

The predicted variance metric uses the predicted variance of the Gaussian process surrogate as the score of a candidate point. Thus, the adaptively chosen points will be in areas of highest uncertainty according to the Gaussian process model.

distance

- Keywords Area
- method
- adaptive_sampling
- fitness_metric
- distance

Space filling metric

Specification

Alias: none
Argument(s): none
6.2. METHOD

Description
The distance metric calculates the Euclidean distance in domain space between the candidate and its nearest neighbor in the set of points already evaluated on the true model. Therefore, the most undersampled area of the domain will always be selected. Note that this is a space-filling metric.

gradient

- Keywords Area
- method
- adaptive_sampling
- fitness_metric
- gradient

Fill the range space of the surrogate

Specification
Alias: none
Argument(s): none

Description
The gradient metric calculates the score as the absolute value of the difference in range space (the outputs) of the two points. The output values used are predicted from the surrogate model. This method attempts to evenly fill the range space of the surrogate.

batch_selection

- Keywords Area
- method
- adaptive_sampling
- batch_selection

(Experimental) How to select new points

Specification
Alias: none
Argument(s): none
Default: naive
## Description

The adaptive sampling is an experimental capability that is not ready for production use at this time.

With batch or multi-point selection, the true model can be evaluated in parallel and thus increase throughput before refitting our surrogate model. This proposes a new challenge as the problem of choosing a single point and choosing multiple points off a surrogate are fundamentally different. Selecting the $n$ best scoring candidates is more than likely to generate a set of points clustered in one area which will not be conducive to adapting the surrogate.

We have implemented several strategies for batch selection of points. These are described in the User’s manual and are the subject of active research.

The batch selection strategies include:

1. naive:
2. distance_penalty
3. constant_liar
4. topology

### naive

- Keywords Area
- method
- adaptive_sampling
- batch_selection
6.2. METHOD

- naive

Take the highest scoring candidates

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

This strategy will select the \( n \) highest scoring candidates regardless of their position. This tends to group an entire round of points in the same area.

**distance_penalty**

- **Keywords Area**
- **method**
- **adaptive_sampling**
- **batch_selection**
- **distance_penalty**

Add a penalty to spread out the points in the batch

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

In this strategy, the highest scoring candidate is selected and then all remaining candidates are re-scored with a distance penalization factor added in to the score.

**topology**

- **Keywords Area**
- **method**
- **adaptive_sampling**
- **batch_selection**
- **topology**

In this selection strategy, we use information about the topology of the space from the Morse-Smale complex to identify next points to select.
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): none

Description

In this strategy we look at the topology of the scoring function and select the $n$ highest maxima in the topology. To determine local maxima, we construct the approximate Morse-Smale complex. This strategy does require the user to have the Morse-Smale package.

count_lia

- Keywords Area
- method
- adaptive_sampling
- batch_selection
- count_lia

Use information from the existing surrogate model to predict what the surrogate upgrade will be with new points.

Specification

Alias: none
  Argument(s): none

Description

The strategy first selects the highest scoring candidate, and then refits the surrogate using a "lie" value at the point selected. The 'lie' value is based on the surrogate predictions and not the simulation. This process repeats until $n$ points have been selected whereupon the lie values are removed from the surrogate and the selected points are evaluated on the true model and the surrogate is refit with these values.

batch_size

- Keywords Area
- method
- adaptive_sampling
- batch_size

The number of points to add in each batch.

Specification

Alias: none
  Argument(s): INTEGER
  Default: 1
6.2. METHOD

Description
The number of points to add in each batch.

import_build_points_file

- Keywords Area
- method
- adaptive_sampling
- import_build_points_file

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file
Argument(s): STRING
Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td></td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

Description
The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior
Be default, methods do not import points from a file.

Usage Tips
Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
• custom.annotated
• freeform

**Examples**

```plaintext
method
    polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

**annotated**

• **Keywords Area**
• **method**
• **adaptive_sampling**
• **import_build_points_file**
• **annotated**

Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

• **file_formats**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format

**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The *annotated* keyword can be used to explicitly specify this.

**Usage Tips**

• To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify `custom.annotated header eval_id`

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated format`, although `freeform` remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.
Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```python
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

custom_annotated

- Keywords Area
- method
- adaptive_sampling
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format
## Required/Optional

<table>
<thead>
<tr>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

## Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```yaml
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.2. METHOD

header

- Keywords Area
- method
- adaptive_sampling
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none
Default: no header

Description

See description of parent custom_annotated

eval_id

- Keywords Area
- method
- adaptive_sampling
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none
Default: no eval_id column

Description

See description of parent custom_annotated
interface_id

- Keywords Area
- method
- adaptive_sampling
- import_build_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent custom.annotated

**freeform**

- Keywords Area
- method
- adaptive_sampling
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format
6.2. METHOD

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
evironment
    tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
0.9    1.1    0.0002    0.26    0.76
0.90009 1.1 0.00019996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

active_only

- Keywords Area
- method
- adaptive_sampling
- import_build_points_file
- active_only

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- file_formats
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword active_only indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

export_approx_points_file

- Keywords Area
- method
- adaptive_sampling
- export_approx_points_file

Output file for evaluations of a surrogate model

Specification

Alias: export_points_file

Argument(s): STRING

Default: no point export to a file

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional-</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>custom_annotated</td>
<td></td>
<td>Selects custom-annotated tabular file format</td>
<td></td>
</tr>
<tr>
<td>freeform</td>
<td></td>
<td>Selects freeform file format</td>
<td></td>
</tr>
</tbody>
</table>

Description

The export_approx_points_file keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:
6.2. METHOD

- annotated (default)
- custom.annotated
- freeform

annotated
- Keywords Area
- method
- adaptive_sampling
- export_approx_points_file
- annotated

Selects annotated tabular file format

Topics
This keyword is related to the topics:
- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips
- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom.annotated header eval_id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```python
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

custom_annotated

- **Keywords Area**
- **method**
- **adaptive_sampling**
- **export_approx_points_file**
- **custom_annotated**

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:

- **file_formats**

Specification

Alias: none

Argument(s): none

Default: annotated format

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>header</td>
<td></td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>eval_id</th>
<th>Enable evaluation ID column in custom-annotated tabular file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
  tabular_data_file = 'dakota_summary.dat'
  custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id  x1  x2  obj_fn  nln_ineq_con_1  nln_ineq_con_2
1 0.9  1.1  0.0002  0.26   0.76
2 0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3 0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

*header*

- Keywords Area
- method
- adaptive_sampling
CHAPTER 6. KEYWORDS AREA

- `export_approx_points_file`
- `custom_annotated`
- `header`

Enable header row in custom-annotated tabular file

**Specification**

Alias: none

- **Argument(s):** none
- **Default:** no header

**Description**

See description of parent `custom_annotated`

- `eval_id`

  - **Keywords Area**
  - **method**
  - `adaptive_sampling`
  - `export_approx_points_file`
  - `custom_annotated`
  - `eval_id`

Enable evaluation ID column in custom-annotated tabular file

**Specification**

Alias: none

- **Argument(s):** none
- **Default:** no `eval_id` column

**Description**

See description of parent `custom_annotated`

- `interface_id`

  - **Keywords Area**
  - **method**
  - `adaptive_sampling`
  - `export_approx_points_file`
  - `custom_annotated`
  - `interface_id`

Enable interface ID column in custom-annotated tabular file
6.2. METHOD

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent custom.annotated

freeform

- Keywords Area
- method
- adaptive_sampling
- export.approx_points_file
- freeform

Selects freeform file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
  tabular_data_file = ’dakota_summary.dat’
  freeform
```

Resulting tabular file:

```
0.9     1.1     0.0002     0.26     0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

`response_levels`

- Keywords Area
- method
- adaptive_sampling
- response_levels

Values at which to estimate desired statistics for each response

Specification

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF probabilities/reliabilities to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_response_ - levels</td>
<td><strong>num_response_levels</strong></td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td>compute</td>
<td><strong>compute</strong></td>
<td>Selection of statistics to compute at each response level</td>
</tr>
</tbody>
</table>
6.2. METHOD

Description

The response_levels specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If response_levels are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If response_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The num_response_levels is used to specify which arguments of the response_level correspond to which response.

Examples

For example, specifying a response_level of 52.3 followed with compute probabilities will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the num_response_levels key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = 'TF1w' 'TF2w'
```
CHAPTER 6. KEYWORDS AREA

histogram_bin_uncertain = 2  
num_pairs = 3 4  
abscissas = 5 8 10 .1 .2 .3 .4  
counts = 17 21 0 12 24 12 0  
descriptors = 'TF1h' 'TF2h'  

histogram_point_uncertain  
real = 1  
num_pairs = 2  
abscissas = 3 4  
counts = 1 1  
descriptors = 'TF3h'  

interface,  
  system asynch evaluation_concurrency = 5  
  analysis_driver = 'text_book'  

responses,  
  response_functions = 3  
  no_gradients  
  no_hessians  

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:  

PDF for response_fn_1:  
Bin Lower Bin Upper Density Value  
--------- --------- -------------  
2.7604749078e+11 3.6000000000e+11 5.3601733194e-12  
3.6000000000e+11 4.0000000000e+11 4.2500000000e-12  
4.0000000000e+11 4.4000000000e+11 3.7500000000e-12  
4.4000000000e+11 5.4196114379e+11 2.2557612778e-12  

PDF for response_fn_2:  
Bin Lower Bin Upper Density Value  
--------- --------- -------------  
4.6431154744e+04 6.0000000000e+04 2.8742313192e-05  
6.0000000000e+04 6.5000000000e+04 6.4000000000e-05  
6.5000000000e+04 7.0000000000e+04 4.0000000000e-05  
7.0000000000e+04 7.8702465755e+04 1.0341896485e-05  

PDF for response_fn_3:  
Bin Lower Bin Upper Density Value  
--------- --------- -------------  
2.3796737090e+05 3.5000000000e+05 4.284660868e-06  
3.5000000000e+05 4.0000000000e+05 8.6000000000e-06  
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06  

Level mappings for each response function:  

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:  
Response Level Probability Level Reliability Index General Rel Index  
---------------------------------  
3.6000000000e+11 5.5000000000e-01  
4.0000000000e+11 3.8000000000e-01  
4.4000000000e+11 2.3000000000e-01  

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:  
Response Level Probability Level Reliability Index General Rel Index  
---------------------------------  
6.0000000000e+04 6.1000000000e-01  
6.5000000000e+04 2.9000000000e-01  
7.0000000000e+04 9.0000000000e-02  

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:  
Response Level Probability Level Reliability Index General Rel Index  
---------------------------------
6.2. METHOD

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

num_response_levels

- Keywords Area
- method
- adaptive_sampling
- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The num_response_levels keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If num_response_levels is not specified, the response_levels will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.
Examples

```plaintext
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

**compute**

- **Keywords Area**
- **method**
- **adaptive_sampling**
- **response_levels**
- **compute**

Selection of statistics to compute at each response level

### Specification

**Alias:** none

**Argument(s):** none

**Default:** probabilities

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>probabilities</strong></td>
<td>Computes probabilities associated with response levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>gen_reliabilities</strong></td>
<td>Computes generalized reliabilities associated with response levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>system</strong></td>
<td>Compute system reliability (series or parallel)</td>
</tr>
</tbody>
</table>


Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen.-reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method sampling
  sample_type random
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
  compute reliabilities

  probabilities

• Keywords Area
• method
• adaptive_sampling
• response_levels
• compute
• probabilities

Computes probabilities associated with response levels
```

Specification

Alias: none

Argument(s): none
Description

The probabilities keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If response_levels is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the probabilities keyword should be specified in conjunction with the compute keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method sampling
  sample_type random
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
  compute probabilities
```

gen_reliabilities

- Keywords Area
- method
- adaptive_sampling
- response_levels
- compute
- gen_reliabilities

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The gen_reliabilities keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If response_levels is specified, the generalized reliabilities are not computed by default. To change this behavior, the gen_reliabilities keyword should be specified in conjunction with the compute keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.
Examples

method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities

system

• Keywords Area

• method

• adaptive_sampling

• response_levels

• compute

• system

Compute system reliability (series or parallel)

Specification

Alias: none
Argument(s): none

| Required/-  | Description of          | Dakota Keyword       | Dakota Keyword          |
| Optional    | Group                  |                     | Description             |
| Required(Choose One) | Group 1 |                     |                        |
| series       |                        |                     | Aggregate response      |
| parallel     |                        |                     | statistics assuming a  |
|              |                        |                     | series system           |
|              |                        |                     | Aggregate response      |
|              |                        |                     | statistics assuming a   |
|              |                        |                     | parallel system         |

Description

With the system probability/reliability option, statistics for specified response_levels are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.
series

- Keywords Area
- method
- adaptive_sampling
- response_levels
- compute
- system
- series

Aggregate response statistics assuming a series system

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent keyword `system` for description.

parallel

- Keywords Area
- method
- adaptive_sampling
- response_levels
- compute
- system
- parallel

Aggregate response statistics assuming a parallel system

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent keyword `system` for description.
6.2. METHOD

**misc_options**

- Keywords Area
- method
- adaptive_sampling
- misc_options

(Experimental) This is a capability used to send the adaptive sampling algorithm specific options.

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** no misc options

**Description**

The adaptive sampling algorithm is an experimental capability and not ready for production use at this time.

**max_iterations**

- Keywords Area
- method
- adaptive_sampling
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient-global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also max_function_evaluations.
CHAPTER 6. KEYWORDS AREA

distribution

- Keywords Area
- method
- adaptive_sampling
- distribution

Selection of cumulative or complementary cumulative functions

Specification

Alias: none
  Argument(s): none
  Default: cumulative (CDF)

<table>
<thead>
<tr>
<th>Requirement/Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>

Description

The distribution keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the distribution keyword is present, it must be accompanied by either cumulative or complementary. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

method
  sampling
  sample_type lhs
  samples = 10
  distribution cumulative
6.2. METHOD

### cumulative

- Keywords Area
- method
- adaptive_sampling
- distribution
- cumulative

Computes statistics according to cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

**Examples**

```dakota
method sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```

### complementary

- Keywords Area
- method
- adaptive_sampling
- distribution
- complementary

Computes statistics according to complementary cumulative functions
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must appear in conjunction with the distribution keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary

probability_levels
```

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none
Argument(s): REALLIST
Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_probability_levels</td>
<td></td>
<td>Specify which probability_levels correspond to which response</td>
</tr>
</tbody>
</table>
Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
      1. 8 .9 0.
      1. 3 .2 0.
variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = 'TF1h' 'TF2h'
histogram_point_uncertain
  real = 1
  num_pairs = 2
  abscissas = 3 4
  counts = 1 1
  descriptors = 'TF3h'
interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'
responses,
  response_functions = 3
```
CHAPTER 6. KEYWORDS AREA

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0634975300e+11</td>
<td>5.1454122311e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.4196114379e+11</td>
<td>2.4334239039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.6997214153e+05</td>
<td>5.302836523e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.8100966235e+05</td>
<td>9.060055634e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.411498127e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td></td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.0000000000e-01</td>
<td>6.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>3.0000000000e-01</td>
<td>3.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>5.4196114379e+11</td>
<td>0.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.0000000000e-01</td>
<td>8.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>5.0000000000e-01</td>
<td>5.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>7.8702465755e+04</td>
<td>0.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td></td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.0000000000e-01</td>
<td>3.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>2.0000000000e-01</td>
<td>2.0000000000e-01</td>
<td></td>
</tr>
<tr>
<td>4.411498127e+05</td>
<td>0.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).
6.2. METHOD

**num_probability_levels**
- Keywords Area
- method
- adaptive_sampling
- probability_levels
- num_probability_levels

Specify which `probability_levels` correspond to which response

**Specification**
- **Alias:** none
  - **Argument(s):** INTEGERLIST
  - **Default:** `probability_levels` evenly distributed among response functions

**Description**
See parent page

**gen_reliability_levels**
- Keywords Area
- method
- adaptive_sampling
- gen_reliability_levels

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**
- **Alias:** none
  - **Argument(s):** REALLIST
  - **Default:** No CDF/CCDF response levels to compute

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional- | Group | num_gen_reliability_levels | Description |
| Optional | | | Specify which `gen_reliability_levels` correspond to which response |

**Description**
Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).
**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_gen_reliability_levels**

- **Keywords Area**
- **method**
- **adaptive_sampling**
- **gen_reliability_levels**
- **num_gen_reliability_levels**

Specify which `gen_reliability_levels` correspond to which response

**Specification**

**Alias:** none

**Argument(s):** INTEGERS

**Default:** `gen_reliability_levels` evenly distributed among response functions

**Description**

See parent page

**rng**

- **Keywords Area**
- **method**
- **adaptive_sampling**
- **rng**

Selection of a random number generator

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** Mersenne twister ( mt19937 )
### Description

The **rng** keyword is used to indicate a choice of random number generator.

#### Default Behavior

If specified, the **rng** keyword must be accompanied by either **rnum2** (pseudo-random numbers) or **mt19937** (random numbers generated by the Mersenne twister). Otherwise, **mt19937**, the Mersenne twister is used by default.

#### Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

### Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

**mt19937**

- **Keywords Area**
- **method**
- **adaptive_sampling**
- **rng**
- **mt19937**

  Generates random numbers using the Mersenne twister

### Specification

**Alias**: none

**Argument(s)**: none
CHAPTER 6. KEYWORDS AREA

Description

The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended.

Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937

rnum2

- Keywords Area
- method
- adaptive_sampling
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
samples

- Keywords Area
- method
- adaptive_sampling
- samples

Number of samples for sampling-based methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be N*(dim+2).

**Examples**

```dakota
method
  sampling
    sample_type lhs
    samples = 20

seed

- Keywords Area
- method
- adaptive_sampling
- seed

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)
CHAPTER 6. KEYWORDS AREA

Description
The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**
If not specified, the seed is randomly generated.

**Expected Output**
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples
```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**model_pointer**
- Keywords Area
- method
- adaptive_sampling
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:
- block_pointer

Specification
**Alias:** none
**Argument(s):** STRING
**Default:** method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.
6.2. METHOD

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.
See `block_pointer` for details about pointers.

Examples

```plaintext
environment
    tabular_graphics_data
    method_pointer = 'UQ'

method
    id_method = 'UQ'
    model_pointer = 'Surr'
    sampling,
        samples = 10
        seed = 98765 rng rnum2
        response_levels = 0.1 0.2 0.6
        0.1 0.2 0.6
        0.1 0.2 0.6
    sample_type lhs
distribution cumulative

model
    id_model = 'Surr'
    surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
    id_method = 'DACE'
    model_pointer = 'DACE_M'
    sampling sample_type lhs
        samples = 121 seed = 5034 rng rnum2

model
    id_model = 'DACE_M'
    single
    interface_pointer = 'I1'

variables
    uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
    response_functions = 3
    no_gradients
    no_hessians
```

6.2.52 pof_darts

- Keywords Area
Probability-of-Failure (POF) darts is a novel method for estimating the probability of failure based on random sphere-packing.

**Topics**
This keyword is related to the topics:
- uncertainty_quantification

**Specification**

Alias: nond_pof_darts
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>lipschitz</td>
<td>lipschitz</td>
<td>Select the type of Lipschitz estimation (global or local)</td>
</tr>
<tr>
<td>Optional</td>
<td>emulator_samples</td>
<td>emulator_samples</td>
<td>Specify the number of samples taken on the emulator to estimate the Probability of Failure in POF Darts</td>
</tr>
<tr>
<td>Optional</td>
<td>response_levels</td>
<td>response_levels</td>
<td>Values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td>distribution</td>
<td>distribution</td>
<td>Selection of cumulative or complementary cumulative functions</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>probability_levels</th>
<th>Specify probability levels at which to estimate the corresponding response value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>gen_reliability_levels</td>
<td>Specify generalized reliability levels at which to estimate the corresponding response value</td>
</tr>
<tr>
<td>Optional</td>
<td>rng</td>
<td>Selection of a random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td>samples</td>
<td>Number of samples for sampling-based methods</td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

*pof_darts* is a novel method for estimating the probability of failure based on random sphere-packing. Random spheres are sampled from the domain with the constraint that each new sphere center has to be outside prior disks. The radius of each sphere is chosen such that the entire sphere lie either in the failure or the non-failure region. This radius depends of the function evaluation at the disk center, the failure threshold and an estimate of the function gradient at the disk center.

We utilize a global surrogate for evaluating the gradient and hence only one function evaluation is required for each sphere.

After exhausting the sampling budget specified by `samples`, which is the number of spheres per failure threshold, the domain is decomposed into two regions. These regions correspond to failure and non-failure, each represented by the union of the spheres of each type. The volume of the union of failure spheres gives a lower bound on the required estimate of the probability of failure, while the volume of the union of the non-failure spheres subtracted from the volume of the domain gives an upper estimate. We currently report the average of both estimates.

*pof_darts* handles multiple response functions and allows each to have multiple failure thresholds. For each failure threshold, *pof_darts* will insert a number of spheres specified by the user-input parameter `samples`.

However, estimating the probability of failure for each failure threshold would utilize the total number of disks sampled for all failure thresholds. For each failure threshold, the sphere radii changes to generate the right spatial decomposition.
lipschitz

- Keywords Area
- method
- pof_darts
- lipschitz

Select the type of Lipschitz estimation (global or local)

**Specification**

**Alias:** none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword local</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>global</td>
<td>Specify global estimation of the Lipschitz estimate</td>
</tr>
</tbody>
</table>

**Description**

There are two types of Lipschitz estimation used in sizing the disks used in POF Darts: **global** and **local**. The global approach uses one Lipschitz estimate for the entire domain. The local approach calculates the Lipschitz estimate separately for each Voronoi region based on nearby points. The local method is more expensive but more accurate.

**local**

- Keywords Area
- method
- pof_darts
- lipschitz
- local

Specify local estimation of the Lipschitz constant

**Specification**

**Alias:** none

Argument(s): none
6.2. **METHOD**

**Description**

The local approach to estimate the Lipschitz constant calculates the Lipschitz estimate separately for each Voronoi region based on nearby points. The local method is more expensive but results in higher accuracy compared to the global method.

**global**

- **Keywords Area**
- **method**
- **pof_darts**
- **lipschitz**
- **global**

Specify global estimation of the Lipschitz estimate

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The global approach uses one Lipschitz estimate for the entire domain. This option is currently deactivated. Please refer to the local alternative.

**emulator_samples**

- **Keywords Area**
- **method**
- **pof_darts**
- **emulator_samples**

Specify the number of samples taken on the emulator to estimate the Probability of Failure in POF Darts

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

The last step of the POF Darts method involves constructing an emulator over the points identified thus far, and sampling that emulator extensively to estimate the probability of failure. **emulator_samples** allows one to specify the number of samples taken on the emulator. The default is 1E+6. If the probability of failure estimate is zero, the user may want to increase the number of emulator samples.
CHAPTER 6. KEYWORDS AREA

response_levels

- Keywords Area
- method
- pof_darts
- response_levels

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_response_levels</td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td>compute</td>
<td>Selection of statistics to compute at each response level</td>
</tr>
</tbody>
</table>

Description

The response_levels specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If response_levels are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If response_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The num_response_levels is used to specify which arguments of the response_level correspond to which response.
6.2. METHOD

Examples

For example, specifying a response level of 52.3 followed with compute probabilities will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the num_response_levels key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                  6.0e+04 6.5e+04 7.0e+04
                  3.5e+05 4.0e+05 4.5e+05
variables,
  normal_uncertain = 2
  means = 248.89, 593.33
  std_deviations = 12.4, 29.7
  descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds = 199.3, 474.63
  upper_bounds = 298.5, 712.
  descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas = 12., 30.
  betas = 250., 590.
  descriptors = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
    num_pairs = 2
    abscissas = 3 4
    counts = 1 1
    descriptors = 'TF3h'
interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'
responses,
  response_functions = 3
  no_gradients
  no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.
Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.360173194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.0341896485e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
<tr>
<td>3.5000000000e+05</td>
<td>4.0000000000e+05</td>
<td>8.6000000000e-06</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>4.5000000000e+05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e-01</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e-01</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e-01</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e-01</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e-02</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td>1.0000000000e-01</td>
<td>1.0000000000e-01</td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

num_response_levels

- Keywords Area
- method
- pof_darts
6.2. METHOD

- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The num_response_levels keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If num_response_levels is not specified, the response_levels will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```plaintext
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5

compute

  Keywords Area
  method
  pof_darts
  response_levels
  compute

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities
### Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

**Default Behavior**

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

**Expected Output**

The type of statistics specified by `compute` will be reported for each response level.

**Usage Tips**

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

### Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
  compute reliabilities
```

### Keywords Area

**Required/Optional Required**

**Description of Group**

**Dakota Keyword**

**Dakota Keyword Description**

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>gen_reliabilities</td>
<td>Computes generalized reliabilities associated with response levels</td>
</tr>
<tr>
<td>Optional</td>
<td>system</td>
<td></td>
<td>Compute system reliability (series or parallel)</td>
</tr>
</tbody>
</table>

- **Keywords Area**
- **method**
- **pof_darts**
6.2. METHOD

- response_levels
- compute
- probabilities

Computes probabilities associated with response levels

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

**Examples**

```
method sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
        6.0e+04 6.5e+04 7.0e+04
        3.5e+05 4.0e+05 4.5e+05
    compute probabilities

gen_reliabilities
```

**Keywords Area**

- method
- pof_darts
- response_levels
- compute
- gen_reliabilities

Computes generalized reliabilities associated with response levels
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```plaintext
method	sampling
    sample_type random
    samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
    6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
compute gen_reliabilities

system

• Keywords Area
• method
• pof_darts
• response_levels
• compute
• system

Compute system reliability (series or parallel)
```

Specification

Alias: none
Argument(s): none
### Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

```
series

- Keywords Area
- method
- pof_darts
- response_levels
- compute
- system
- series
```

Aggregate response statistics assuming a series system

### Specification

**Alias:** none

**Argument(s):** none

### Description

See parent keyword `system` for description.
parallel

- Keywords Area
- method
- pof_darts
- response_levels
- compute
- system
- parallel

Aggregate response statistics assuming a parallel system

**Specification**

*Alias:* none

*Argument(s):* none

**Description**

See parent keyword `system` for description.

distribution

- Keywords Area
- method
- pof_darts
- distribution

Selection of cumulative or complementary cumulative functions

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* cumulative (CDF)

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th>Group 1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Choose One</em></td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
</tbody>
</table>
6.2. METHOD

| complementary | Computes statistics according to complementary cumulative functions |

### Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

#### Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

#### Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

### Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
  distribution cumulative
```

### Specification

**Alias:** none

**Argument(s):** none

### Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

#### Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.
Expected Outputs
Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

complementary

- Keywords Area
- method
- pof_darts
- distribution
- complementary

Computes statistics according to complementary cumulative functions

Specification

Alias: none
Argument(s): none

Description
Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior
By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must be appear in conjunction with the distribution keyword.

Expected Outputs
Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```
6.2. METHOD

**probability_levels**

- **Keywords Area**
  - method
  - pof_darts
  - probability_levels

Specify probability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>num_probability_levels</td>
<td>Specify which probability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

**Expected Output**

If **probability_levels** are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the **probability_levels** and/or **response_levels** in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

**Examples**

The Dakota input file below specifies a sampling method with probability levels of interest.

```plaintext
method, sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1.66 0.33 0.1 .8 .5 0.1 .3 .2 0.
variables,
  normal_uncertain = 2
  means = 248.89, 593.33
  std_deviations = 12.4, 29.7
```
CHAPTER 6. KEYWORDS AREA

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated.
Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0634975300e+11</td>
<td>5.1454122311e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.4196114379e+11</td>
<td>2.4334239039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.891610839e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796773709e+05</td>
<td>3.6997214153e+05</td>
<td>5.3028836523e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.8100966235e+05</td>
<td>9.060055634e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.4111498127e+05</td>
<td>3.2274925348e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.0000000000e-01</td>
<td>3.6000000000e-01</td>
<td>1.8000000000e-01</td>
</tr>
<tr>
<td>4.6431154744e+11</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
<td>1.0000000000e+00</td>
</tr>
</tbody>
</table>
6.2. METHOD

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.643154744e+04</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.651182775e+04</td>
<td>8.0000000000e+01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.160381379e+04</td>
<td>5.0000000000e+01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.870246555e+04</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.379673709e+05</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.699721415e+05</td>
<td>3.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.810096623e+05</td>
<td>2.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.411149812e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num.probability.levels

- Keywords Area
- method
- pof.darts
- probability.levels
- num.probability.levels

Specify which probability.levels correspond to which response

Specification

Alias: none
Argument(s): INTEGERLIST
Default: probability.levels evenly distributed among response functions

Description

See parent page
**CHAPTER 6. KEYWORDS AREA**

**gen_reliability_levels**
- **Keywords Area**
- **method**
- **pof_darts**
- **gen_reliability_levels**

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>num_gen_reliability_levels</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>Specify which</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>gen_reliability_levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>correspond to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_gen_reliability_levels**
- **Keywords Area**
- **method**
- **pof_darts**
- **gen_reliability_levels**
- **num_gen_reliability_levels**

Specify which **gen_reliability_levels** correspond to which response
6.2. METHOD

Specification

Alias: none

Argument(s): INTEGERLIST
Default: gen_reliability_levels evenly distributed among response functions

Description

See parent page

rng

- Keywords Area
- method
- pof_darts
- rng

Selection of a random number generator

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword mt19937</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>rnum2</td>
<td>Generates pseudo-random numbers using the Pecos package</td>
</tr>
</tbody>
</table>

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.
CHAPTER 6. KEYWORDS AREA

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

`mt19937`

- Keywords Area
- method
- pof_darts
- rng
- mt19937

Generates random numbers using the Mersenne twister

Specification

Alias: none
Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

`rnum2`

- Keywords Area
- method
- pof_darts
- rng
6.2. METHOD

- rnum2

Generates pseudo-random numbers using the Pecos package

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

**Default Behavior**

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

**Examples**

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

```
samples
```

- **Keywords Area**
- **method**
- **pof_darts**
- **samples**

Number of samples for sampling-based methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**
To obtain linear sensitivities or to construct a linear response surface, at least \( \text{dim}+1 \) samples should be used, where \( \text{dim} \) is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least \( \frac{(\text{dim}+1)(\text{dim}+2)}{2} \) samples are needed. For uncertainty quantification, we recommend at least \( 10+\text{dim} \) samples. For \texttt{variance\_based\_decomp}, we recommend hundreds to thousands of samples. Note that for \texttt{variance\_based\_decomp}, the number of simulations performed will be \( N(\text{dim}+2) \).


**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 20

seed
  - Keywords Area
  - method
  - pof_darts
  - seed

Seed of the random number generator
```

**Specification**

Alias: none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)

**Description**

The random \texttt{seed} control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If \texttt{seed} is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without \texttt{seed} specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```
6.2. METHOD

model_pointer

- Keywords Area
- method
- pof_darts
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                  0.1 0.2 0.6
                  0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.53  rkd_darts
  
  • Keywords Area
  
  • method
  
  • rkd_darts

Recursive k-d (RKD) Darts: Recursive Hyperplane Sampling for Numerical Integration of High-Dimensional Functions.

Topics
This keyword is related to the topics:

  • uncertainty_quantification

Specification
Alias: nond_rkd_darts

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
</table>


| Optional | lipschitz | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. Specify the number of samples taken on the emulator to estimate the Numerical Integration in RKD Darts. |
| Optional | emulator_samples | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
| Optional | response_levels | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
| Optional | distribution | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
| Optional | probability_levels | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
| Optional | gen_reliability_levels | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
| Optional | rng | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
## Description

Disclaimer: The RKD method is currently in development mode, for further experimental verification. Please contact Dakota team if you have further questions about using this method.

Recursive k-d (RKD) darts is an algorithm to evaluate the integration of a d-dimensional black box function \( f(x) \) via recursive sampling over d, using a series of hyperplanes of variable dimensionality \( k = \{d, d-1, \ldots, 0\} \). Fundamentally, we decompose the d-dimensional integration problem into a series of nested one-dimensional problems. That is, we start at the root level (the whole domain) and start sampling down using hyperplanes of one lower dimension, all the way down to zero (points). A d-dimensional domain is subsampled using (d-1) hyperplanes, a (d-1)-dimensional sub-domain is subsampled using (d-2) hyperplanes, and so on. Every hyperplane, regardless of its dimension, is evaluated using sampled hyperplanes of one lower dimension, as shown in the set of figures above. Each hyperplane has direct information exchange with its parent hyperplane of one higher dimension, and its children of one lower dimension.

In each one-dimensional problem, we construct a piecewise approximation surrogate model, using 1-dimensional Lagrange interpolation. Information is exchanged between different levels, including integration values, as well as interpolation and evaluation errors, in order to a) find the integration value up to that level, b) estimate the associated integration error, and c) guide the placement of future samples.

### lipschitz

- Keywords Area
- method
- rkd_darts
- lipschitz

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

## Specification

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>samples</td>
<td>samples</td>
<td>Number of</td>
<td>Number of</td>
</tr>
<tr>
<td>seed</td>
<td>seed</td>
<td>samples for</td>
<td>samples for</td>
</tr>
<tr>
<td>model_pointer</td>
<td>model_pointer</td>
<td>seed of the random number generator</td>
<td>seed of the random number generator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identifier for model block to be used by a method</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>
6.2. METHOD

Required *(Choose One)*

<table>
<thead>
<tr>
<th>Group 1</th>
<th>local</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group 1</th>
<th>global</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
</tbody>
</table>

local

- Keywords Area
- method
- rkd_darts
- lipschitz
- local

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none
Argument(s): none

global

- Keywords Area
- method
- rkd_darts
- lipschitz
- global

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none
Argument(s): none
emulator_samples

- Keywords Area
- method
- rkd_darts
- emulator_samples

Specify the number of samples taken on the emulator to estimate the Numerical Integration in RKD Darts.

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

Disclaimer: The RKD method is currently in development mode, for further experimental verification. Please contact Dakota team if you have further questions about using this method.

The last step of the RKD Darts method involves constructing an emulator over the points identified thus far, and sampling that emulator extensively to estimate the value of the numerical integration. `emulator_samples` allows one to specify the number of samples taken on the emulator. The default is 1E+6.

**response_levels**

- Keywords Area
- method
- rkd_darts
- response_levels

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF probabilities/reliabilities to compute

| Required/- | Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional | | num_response_-levels | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
| Optional | | compute | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |
num_response_levels

- Keywords Area
- method
- rkd_darts
- response_levels
- num_response_levels

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** response_levels evenly distributed among response functions

**compute**

- Keywords Area
- method
- rkd_darts
- response_levels
- compute

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** probabilities

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword probabilities</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>gen_reliabilities</td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>system</td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
</tbody>
</table>
probabilities

• Keywords Area
• method
• rkd_darts
• response_levels
• compute
• probabilities

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

**Specification**

**Alias:** none

**Argument(s):** none

gen_reliabilities

• Keywords Area
• method
• rkd_darts
• response_levels
• compute
• gen_reliabilities

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

**Specification**

**Alias:** none

**Argument(s):** none

system

• Keywords Area
• method
• rkd_darts
• response_levels
• compute
• system

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
6.2. METHOD

Specification

Alias: none
  Argument(s): none
### CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required/-Optional Required</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword series</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>Required</td>
<td>series</td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
<tr>
<td>optional</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alias:</strong> none</td>
</tr>
<tr>
<td><strong>Argument(s):</strong> none</td>
</tr>
</tbody>
</table>

#### series
- Keywords Area
- method
- rkd_darts
- response_levels
- compute
- system
- series

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

### Specification

**Alias:** none

**Argument(s):** none

#### parallel
- Keywords Area
- method
- rkd_darts
- response_levels
- compute
- system
- parallel

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
6.2. METHOD

Specification

Alias: none
Argument(s): none

distribution

- Keywords Area
- method
- rkd_darts
- distribution

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose One</td>
<td></td>
<td>cumulative</td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
</tbody>
</table>

cumulative

- Keywords Area
- method
- rkd_darts
- distribution
- cumulative

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none
Argument(s): none
Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

**Specification**

**Alias:** none

**Argument(s):** none

**probability_levels**

- Keywords Area
- method
- rkd_darts
- probability_levels

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_probability_levels</td>
<td></td>
<td></td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
</tbody>
</table>

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
6.2. METHOD

Specification

Alias: none

Argument(s): INTEGERLIST
Default: probability_levels evenly distributed among response functions

gen_reliability_levels

- Keywords Area
- method
- rkd_darts
- gen_reliability_levels

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none

Argument(s): REALLIST
Default: No CDF/CCDF response levels to compute

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional   | Group          | num_gen_reliability_levels | num_gen_reliability_levels |
| Optional   |               | num_gen_reliability_levels | Undocumented: Recursive k-d (RKD) Darts is an experimental capability. |

num_gen_reliability_levels

- Keywords Area
- method
- rkd_darts
- gen_reliability_levels
- num_gen_reliability_levels

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none

Argument(s): INTEGERLIST
Default: gen_reliability_levels evenly distributed among response functions
CHAPTER 6. KEYWORDS AREA

rng

- Keywords Area
- method
- rkd_darts
- rng

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none
Argument(s): none
Default: Mersenne twister (mt19937)

| Required/-Optional Required (Choose One) | Description of Group
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mt19937</td>
<td>Description of Group 1</td>
</tr>
<tr>
<td>mt19937</td>
<td>Dakota Keyword</td>
</tr>
<tr>
<td>mt19937</td>
<td>Description</td>
</tr>
<tr>
<td>mt19937</td>
<td>Dakota Keyword</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dakota Keyword</th>
<th>Description of Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>mt19937</td>
<td>Undocumented: Recursive k-d (RKD) Darts is an experimental capability.</td>
</tr>
</tbody>
</table>

rtm2

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none
Argument(s): none
6.2. METHOD

rnum2
- Keywords Area
- method
- rkd_darts
- rng
- num2

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none
Argument(s): none

samples
- Keywords Area
- method
- rkd_darts
- samples

Number of samples for sampling-based methods

Specification

Alias: none
Argument(s): INTEGER
Default: method-dependent

Description

The samples keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior
- By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips
- To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance_based_decomp, we recommend hundreds to thousands of samples. Note that for variance_based_decomp, the number of simulations performed will be N*(dim+2).

Examples

method
  sampling
    sample_type lhs
    samples = 20
seed

- Keywords Area
- method
- rkd_darts
- seed

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)

**Description**

The random *seed* control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If *seed* is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without *seed* specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**model_pointer**

- Keywords Area
- method
- rkd_darts
- model_pointer

Identifier for model block to be used by a method
6.2. METHOD

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                  0.1 0.2 0.6
                  0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single```
interface_pointer = 'I1'

variables
    uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
    response_functions = 3
    no_gradients
    no_hessians

6.2.54 efficient_subspace

• Keywords Area

• method

• efficient_subspace

(Experimental) efficient subspace method (ESM)

Topics
This keyword is related to the topics:

• uncertainty_quantification

Specification

Alias: nond_efficient_subspace

Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>emulator_samples</td>
<td>Number of data points used to train the surrogate model or emulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch_size</td>
<td>The number of points to add in each batch.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
</tbody>
</table>
### Description

ESM is experimental and its implementation is incomplete. It is an active subspace method, intended for use with models with high dimensional input parameter spaces and analytic gradients. The method works by evaluating the response gradient at a number of points in the input parameter space and using a singular value decomposition to identify key linear combinations of input directions along which the response varies. Then UQ is performed in
the reduced input parameter space.

**emulator_samples**

- **Keywords Area**
- **method**
- **efficient_subspace**
- **emulator_samples**

Number of data points used to train the surrogate model or emulator

**Specification**

**Alias**: none  
**Argument(s)**: INTEGER

**Description**

This keyword refers to the number of build points or training points used to construct a Gaussian process emulator. If the user specifies a number of `emulator_samples` that is less than the minimum number of points required to build the GP surrogate, Dakota will augment the samples to obtain the minimum required.

**batch_size**

- **Keywords Area**
- **method**
- **efficient_subspace**
- **batch_size**

The number of points to add in each batch.

**Specification**

**Alias**: none  
**Argument(s)**: INTEGER  
**Default**: 1

**Description**

The number of points to add in each batch.
6.2. METHOD

max_iterations

- Keywords Area
- method
- efficient_subspace
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- efficient_subspace
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4
Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_function_evaluations

- Keywords Area
- method
- efficient_subspace
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also `max_iterations`. 
6.2. METHOD

distribution

- Keywords Area
- method
- efficient_subspace
- distribution

Selection of cumulative or complementary cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>

**Description**

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

**Default Behavior**

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

**Examples**

```plaintext
method sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```
cumulative

- Keywords Area
- method
- efficient_subspace
- distribution
- cumulative

Computes statistics according to cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

**Examples**

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

complementary

- Keywords Area
- method
- efficient_subspace
- distribution
- complementary

Computes statistics according to complementary cumulative functions
6.2. METHOD

Specification

Alias: none
  Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must appear in conjunction with the distribution keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary

probability_levels
  • Keywords Area
  • method
  • efficient_subspace
  • probability_levels
```

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none
  Argument(s): REALLIST
  Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Optional Group</td>
<td>num_probability_</td>
<td>levels</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If probability_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```dakota
method,
  sampling,
  samples = 100 seed = 1
complementary distribution
  probability_levels = 1 .66 .33 0.
      1 .8 .5 0.
      1 .3 .2 0.
variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = ‘TF1n’ ‘TF2n’
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = ‘TF1u’ ‘TF2u’
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = ‘TF1w’ ‘TF2w’
histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = ‘TF1h’ ‘TF2h’
histogram_point_uncertain
  real = 1
  num_pairs = 2
  abscissas = 3 4
  counts = 1 1
  descriptors = ‘TF3h’
interface,
  system async evaluation_concurrency = 5
  analysis_driver = ‘text_book’
responses,
  response_functions = 3
  no_gradients
  no_hessians
```
6.2. METHOD

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0634975300e+11</td>
<td>5.145122311e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.4196114379e+11</td>
<td>2.4334239039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.6997214153e+05</td>
<td>5.3028386523e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.8100966235e+05</td>
<td>9.060055634e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.411498127e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.00000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.60000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>3.30000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4196114379e+11</td>
<td>0.00000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>1.00000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>5.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.8702465755e+04</td>
<td>0.00000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>1.00000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>2.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.411498127e+05</td>
<td>0.00000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_probability_levels
Keywords Area

- method
- efficient_subspace
- probability_levels
- num_probability_levels

Specify which probability_levels correspond to which response

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** probability_levels evenly distributed among response functions

**Description**

See parent page

**gen_reliability_levels**

- Keywords Area
- method
- efficient_subspace
- gen_reliability_levels

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_gen_reliability_levels</td>
<td>gen_reliability_levels</td>
<td>Specify which gen_reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).
6.2. METHOD

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

\texttt{num\_gen\_reliability\_levels}

- Keywords Area
- method
- efficient\_subspace
- gen\_reliability\_levels
- num\_gen\_reliability\_levels

Specify which gen\_reliability\_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen\_reliability\_levels evenly distributed among response functions

Description

See parent page

\texttt{rng}

- Keywords Area
- method
- efficient\_subspace
- rng

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister ( mt19937 )
### Description

The `rng` keyword is used to indicate a choice of random number generator.

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**
The default is recommended, as the Mersenne twister is a higher quality random number generator.

### Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

### `mt19937`

- Keywords Area
- method
- efficient_subspace
- `rng`
- `mt19937`

Generates random numbers using the Mersenne twister

### Specification

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description
The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

Default Behavior
mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

Usage Tips
Use of the Mersenne twister random number generator (mt19937) is recommended.

Examples
```
method sampling
   sample_type lhs
   samples = 10
   seed = 98765
   rng mt19937
```

rnum2
- Keywords Area
- method
- efficient_subspace
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package

Specification
Alias: none
Argument(s): none

Description
The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior
rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

Usage Tips
Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples
```
method sampling
   sample_type lhs
   samples = 10
   seed = 98765
   rng rnum2
```
samples

- Keywords Area
- method
- efficient_subspace
- samples

Number of samples for sampling-based methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where “dim” is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance-based decomp, we recommend hundreds to thousands of samples. Note that for variance-based_decomp, the number of simulations performed will be N*(dim+2).

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 20
```

**seed**

- Keywords Area
- method
- efficient_subspace
- seed

Seed of the random number generator

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)
Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**
If not specified, the seed is randomly generated.

**Expected Output**
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**model_pointer**

- **Keywords Area**
- **method**
- **efficient_subspace**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.
Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.
See block_pointer for details about pointers.

Examples

```
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.55 global_evidence
```
6.2. METHOD

- method
- global_evidence

Evidence theory with evidence measures computed with global optimization methods

Topics

This keyword is related to the topics:

- epistemic_uncertainty_quantification_methods
- evidence_theory

Specification

Alias: nond_global_evidence

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional(Choose One)</td>
<td>Group 1</td>
<td>sbo</td>
<td>Use the surrogate based optimization method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ego</td>
<td>Use the Efficient Global Optimization method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ea</td>
<td>Use an evolutionary algorithm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lhs</td>
<td>Uses Latin Hypercube Sampling (LHS) to sample variables</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>response_levels</td>
<td>Values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>distribution</td>
<td>Selection of cumulative or complementary cumulative functions</td>
</tr>
<tr>
<td>Optional</td>
<td>probability_levels</td>
<td>Specify probability levels at which to estimate the corresponding response value</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>gen_reliability_-</td>
<td>Specify generalized reliability levels at which to estimate the corresponding response value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>levels</td>
<td>Selection of a random number generator</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>rng</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>samples</td>
<td>Number of samples for sampling-based methods</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>Seed of the random number generator</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
<td></td>
</tr>
</tbody>
</table>

Description

global_evidence allows the user to specify several global approaches for calculating the belief and plausibility functions:

- **lhs** - note: this takes the minimum and maximum of the samples as the bounds per "interval cell combination."
- **ego** - uses Efficient Global Optimization which is based on an adaptive Gaussian process surrogate.
- **sbo** - uses a Gaussian process surrogate (non-adaptive) within an optimization process.
- **ea** - uses an evolutionary algorithm. This can be expensive as the ea will be run for each interval cell combination.

Note that to calculate the plausibility and belief cumulative distribution functions, one has to look at all combinations of intervals for the uncertain variables. In terms of implementation, if one is using LHS sampling as outlined above, this method creates a large sample over the response surface, then examines each cell to determine the minimum and maximum sample values within each cell. To do this, one needs to set the number of samples relatively high: the default is 10,000 and we recommend at least that number. If the model you are running is a simulation that is computationally quite expensive, we recommend that you set up a surrogate model within the Dakota input file so that global_evidence performs its sampling and calculations on the surrogate and not on the original model. If one uses optimization methods instead to find the minimum and maximum sample values within each cell, this can also be computationally expensive.
Additional Resources
See the topic page evidence_theory for important background information and usage notes.
Refer to variable_support for information on supported variable types.

Theory
The basic idea is that one specifies an “evidence structure” on uncertain inputs and propagates that to obtain belief and plausibility functions on the response functions. The inputs are defined by sets of intervals and Basic Probability Assignments (BPAs). Evidence propagation is computationally expensive, since the minimum and maximum function value must be calculated for each “interval cell combination.” These bounds are aggregated into belief and plausibility.

See Also
These keywords may also be of interest:

- global_interval_est
- local_evidence
- local_interval_est

sbo

- Keywords Area
- method
- global_evidence
- sbo

Use the surrogate based optimization method

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>gaussian_process</td>
<td>Gaussian Process surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
</tbody>
</table>
### Description

A surrogate-based optimization method will be used. The surrogate employed in sbo is a Gaussian process surrogate.

The main difference between ego and the sbo approach is the objective function being optimized. ego relies on an expected improvement function, while in sbo, the optimization proceeds using an evolutionary algorithm (coliny_ea) on the Gaussian process surrogate: it is a standard surrogate-based optimization. Also note that the sbo option can support optimization over discrete variables (the discrete variables are relaxed) while ego cannot.

This is not the same as surrogate_based_global.

### gaussian_process

- Keywords Area
- method
- global_evidence
- sbo
- gaussian_process

Gaussian Process surrogate model

### Specification

**Alias:** kriging

**Argument(s):** none

**Default:** Surfpack Gaussian process

<table>
<thead>
<tr>
<th>Required-/Optional</th>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>export_approx_points_file</td>
<td>Output file for evaluations of a surrogate model</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>dakota</th>
<th>Select the built in Gaussian Process surrogate</th>
</tr>
</thead>
</table>
6.2. METHOD

Description
Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the surfpack keyword. An alternate version of GP surrogates was available in prior versions of Dakota. For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.

surfpack

• Keywords Area
• method
• global_evidence
• sbo
• gaussian_process
• surfpack

Use the Surfpack version of Gaussian Process surrogates

Specification
Alias: none
Argument(s): none

Description
This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.
Several user options are available:

1. Optimization methods:
   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See optimization_method.
   The total number of evaluations of the likelihood function can be controlled using the max_trials keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:
   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See trend.

3. Correlation Lengths:
   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.
4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The surfpack model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here “inexpensive” means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

dakota

- Keywords Area
- method
- global_evidence
- sbo
- gaussian_process
- dakota

Select the built in Gaussian Process surrogate
6.2. METHOD

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the gaussian_process dakota model can be found in[58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a “nugget,” but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

**use_derivatives**

- Keywords Area
- method
- global_evidence
- sbo
- use_derivatives

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The use_derivatives flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it’s use with Surfpack Gaussian process is not recommended.
import_build_points_file

- Keywords Area
- method
- global_evidence
- sbo
- import_build_points_file

File containing points you wish to use to build a surrogate

### Specification

**Alias:** import_points_file  
**Argument(s):** STRING  
**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom.annotated</td>
<td>selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>selects freeform file format</td>
</tr>
</tbody>
</table>

| Optional | active_only | Import only active variables from tabular data file |

### Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**  
Be default, methods do not import points from a file.

**Usage Tips**  
Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom.annotated
- freeform
6.2. METHOD

Examples

```plaintext
method
  polynomial_chaos
    expansion_order = 4
  import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
  • Keywords Area
  • method
  • global_evidence
  • sbo
  • import_build_points_file
  • annotated

Selects annotated tabular file format
```

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
  Argument(s): none
  Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated_header_eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

`custom_annotated`

- **Keywords Area**
- **method**
- **global_evidence**
- **sbo**
- **import_build_points_file**
- **custom_annotated**

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- **file_formats**

Specification

**Alias:** none
**Argument(s):** none
**Default:** annotated format
6.2. METHOD

### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

#### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

#### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

#### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id    x1    x2    obj_fn  nln_ineq_con_1  nln_ineq_con_2
1   0.9     1.1  0.0002   0.26       0.76
2   0.90009 1.1  0.0001996404857 0.2601620081 0.759955
3   0.89991 1.1  0.0002003604863 0.2598380081 0.760045
...```
CHAPTER 6. KEYWORDS AREA

header
- Keywords Area
- method
- global_evidence
- sbo
- import_build_points_file
- custom.annotated
- header

Enable header row in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no header

Description
See description of parent custom.annotated

eval_id
- Keywords Area
- method
- global_evidence
- sbo
- import_build_points_file
- custom.annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no eval_id column

Description
See description of parent custom.annotated
6.2. METHOD

interface_id

- Keywords Area
- method
- global_evidence
- sbo
- import_build_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent custom_annotated

**freeform**

- Keywords Area
- method
- global_evidence
- sbo
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- fileFormats

**Specification**

Alias: none

**Argument(s):** none

**Default:** annotated format
Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
    tabular_data
      tabular_data_file = ‘dakota_summary.dat’
      freeform
```

Resulting tabular file:

```
0.9  1.1  0.0002  0.26  0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

active_only

• Keywords Area

• method

• global_evidence

• sbo

• import_build_points_file

• active_only

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

• file_formats
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

`export_approx_points_file`

- Keywords Area
  - method
  - global_evidence
  - sbo
  - `export_approx_points_file`

Output file for evaluations of a surrogate model

Specification

Alias: `export_points_file`
Argument(s): STRING
Default: no point export to a file

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Optional/Choose One</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/</td>
<td></td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td>custom_annotated</td>
<td></td>
<td></td>
<td></td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td>freeform</td>
<td></td>
<td></td>
<td></td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

Description

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.
Usage Tips
Dakota exports tabular data in one of three formats:
- annotated (default)
- custom_annotated
- freeform

annotated
- Keywords Area
- method
- global_evidence
- sbo
- export_approx_points_file
- annotated
Selects annotated tabular file format

Topics
This keyword is related to the topics:
- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips
- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval.id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
6.2. METHOD

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

custom_annotated

- Keywords Area
- method
- global_evidence
- sbo
- export_approx_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
tabular_data
tabular_data_file = ’dakota_summary.dat’
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id   x1     x2     obj_fn  nln_ineq_con_1  nln_ineq_con_2
1  0.9  1.1  0.0002  0.26  0.76
2  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...`

header

- Keywords Area
- method
- global_evidence
6.2. METHOD

- sbo
- export_approx_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent custom_annotated

**eval_id**

- Keywords Area
- method
- global_evidence
- sbo
- export_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent custom_annotated
interface_id
  • Keywords Area
  • method
  • global_evidence
  • sbo
  • export_approx_points_file
  • custom_annotated
  • interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* no interface_id column

**Description**

See description of parent custom_annotated

freeform
  • Keywords Area
  • method
  • global_evidence
  • sbo
  • export_approx_points_file
  • freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

  • file_formats

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* annotated format
6.2. METHOD

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
```

ego

• Keywords Area

• method

• global_evidence

• ego

Use the Efficient Global Optimization method

Specification

Alias: none
Argument(s): none
**gaussian.process**

- **Keywords Area**
- **method**
- **global_evidence**
- **ego**
- **gaussian.process**

Gaussian Process surrogate model

**Specification**

Alias: kriging

**Argument(s):** none

**Default:** Surfpack Gaussian process

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>gaussian_process</td>
<td>Gaussian Process surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_-points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>export_approx_-points_file</td>
<td>Output file for evaluations of a surrogate model</td>
</tr>
</tbody>
</table>
6.2. METHOD

### Required (Choose One)

<table>
<thead>
<tr>
<th>Group 1</th>
<th>surfpack</th>
<th>Use the Surfpack version of Gaussian Process surrogates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dakota</td>
<td>Select the built in Gaussian Process surrogate</td>
</tr>
</tbody>
</table>

#### Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the `surfpack` keyword. An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

- **surfpack**
  - Keywords Area
  - method
  - global evidence
  - ego
  - gaussian_process
  - surfpack

Use the Surfpack version of Gaussian Process surrogates

#### Specification

**Alias:** none  
**Argument(s):** none

#### Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. **Optimization methods:**
   
   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See `optimization_method`.
   
   The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the “truth” model, and is relatively inexpensive to compute.

2. **Trend Function:**
   
   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See `trend`. 

3. Correlation Lengths:
   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.

4. Ill-conditioning
   One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

   Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

   The surfpack model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK):
   The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

   See notes in the Theory section.

Theory

Gradient Enhanced Kriging
   Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here “inexpensive” means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

   This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

   dakota
   • Keywords Area
   • method
   • global_evidence
   • ego
   • gaussian_process
   • dakota

Select the built in Gaussian Process surrogate
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the gaussian_process dakota model can be found in[58].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

use_derivatives

- Keywords Area
- method
- global_evidence
- ego
- use_derivatives

Use derivative data to construct surrogate models

Specification

Alias: none
Argument(s): none
Default: use function values only

Description

The use_derivatives flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.
import_build_points_file

- **Keywords Area**
- **method**
- **global_evidence**
- **ego**
- **import_build_points_file**

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file

**Argument(s):** STRING

**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td></td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- **annotated** (default)
- **custom_annotated**
- **freeform**
6.2. METHOD

Examples

```python
method
    polynomial_chaos
        expansion_order = 4
        import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
    • Keywords Area
    • method
    • global_evidence
    • ego
    • import_build_points_file
    • annotated

Selects annotated tabular file format
```

Topics

This keyword is related to the topics:

• file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

• To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

**custom_annotated**

- **Keywords Area**
- **method**
- **global_evidence**
- **ego**
- **import_build_points_file**
- **custom_annotated**

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- **file_formats**

Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated format
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id  x1   x2   obj_fn  nln_ineq_con_1  nln_ineq_con_2
1  0.9   1.1  0.0002  0.26         0.76
2  0.90009  1.1  0.0001996404857 0.2601620081 0.759955
3  0.89991  1.1  0.0002003604863 0.2598380081 0.760045
...```
header
- Keywords Area
- method
- global_evidence
- ego
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent `custom_annotated`

eval_id
- Keywords Area
- method
- global_evidence
- ego
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent `custom_annotated`
6.2. METHOD

interface_id

- Keywords Area
- method
- global_evidence
- ego
- import_build_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent custom.annotated

**freeform**

- Keywords Area
- method
- global_evidence
- ego
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format
CHAPTER 6. KEYWORDS AREA

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```python
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
0.9  1.1  0.0002  0.26  0.76
0.90009  1.1  0.0001996404857  0.2601620081  0.759955
0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

active_only
- Keywords Area
- method
- global_evidence
- ego
- import_build_points_file
- active_only
Import only active variables from tabular data file

Topics
This keyword is related to the topics:
- file_formats
6.2. **METHOD**

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

**export_approx_points_file**

- **Keywords Area**
- **method**
- **global_evidence**
- **ego**
- **export_approx_points_file**

Output file for evaluations of a surrogate model

**Specification**

**Alias:** `export_points_file`

**Argument(s):** STRING

**Default:** no point export to a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Group of Group of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Description</td>
</tr>
<tr>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freeform</td>
<td>Selects freeform file format</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.
Usage Tips
Dakota exports tabular data in one of three formats:

- \texttt{annotated} (default)
- \texttt{custom_annotated}
- \texttt{freeform}

\textbf{annotated}

- \texttt{Keywords Area}
- \texttt{method}
- \texttt{global_evidence}
- \texttt{ego}
- \texttt{export_approx_points_file}
- \texttt{annotated}

Selects annotated tabular file format

Topics
This keyword is related to the topics:

- \texttt{file_formats}

Specification

\textbf{Alias:} none

\textbf{Argument(s):} none

\textbf{Default:} annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

\textbf{Default Behavior}
By default, Dakota imports and exports tabular files in annotated format. The \texttt{annotated} keyword can be used to explicitly specify this.

\textbf{Usage Tips}

- To specify pre-Dakota 6.1 tabular format, which did not include interface ID, specify \texttt{custom_annotated header eval.id}

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to \texttt{annotated format}, though \texttt{freeform} remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the \texttt{freeform} option.
6.2. METHOD

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id  interface  x1  x2  obj_fn  nln_ineq_con_1  nln_ineq_con_2
1  NO_ID  0.9  1.1  0.0002  0.26  0.76
2  NO_ID  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3  NO_ID  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

- **custom_annotated**
  - Keywords Area
  - method
  - global_evidence
  - ego
  - export_approx_points_file
  - custom_annotated
    Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:
- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>header</td>
<td></td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>eval_id</th>
<th>Enable evaluation ID column in custom-annotated tabular file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id    x1    x2    obj_fn  nln_ineq_con_1  nln_ineq_con_2  
1  0.9   1.1   0.0002   0.26     0.76
2  0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3  0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

### header

- Keywords Area
- method
- global_evidence
6.2. METHOD

- ego
- export_approx_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent `custom_annotated`

**eval_id**

- Keywords Area
- method
- global_evidence
- ego
- export_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent `custom_annotated`
interface_id
  • Keywords Area
  • method
  • global_evidence
  • ego
  • export_approx_points_file
  • custom_annotated
  • interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none

  Argument(s): none
  Default: no interface_id column

**Description**

See description of parent custom_annotated

freeform
  • Keywords Area
  • method
  • global_evidence
  • ego
  • export_approx_points_file
  • freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

  • file_formats

**Specification**

Alias: none

  Argument(s): none
  Default: annotated format
Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```python
environment
    tabular_data
        tabular_data_file = 'dakota_summary.dat'
        freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

ea

- Keywords Area
- method
- global_evidence
- ea

Use an evolutionary algorithm

Specification

Alias: none
Argument(s): none
Description

In this approach, the evolutionary algorithm from Coliny, coliny.ea, is used to perform the interval optimization with no surrogate model involved. Again, this option of ea can support interval optimization over discrete variables.

lhs

• Keywords Area
• method
• global_evidence
• lhs

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The lhs keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

By default, Latin Hypercube Sampling is used. To explicitly specify this in the Dakota input file, however, the lhs keyword must appear in conjunction with the sample_type keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

method
  sampling
    sample_type lhs
    samples = 20

response_levels

• Keywords Area
• method
• global_evidence
• response_levels

Values at which to estimate desired statistics for each response
6.2. METHOD

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_response_levels</td>
<td></td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td>compute</td>
<td></td>
<td>Selection of statistics to compute at each response level</td>
</tr>
</tbody>
</table>

Description

The response_levels specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If response_levels are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If response_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The num_response_levels is used to specify which arguments of the response_level correspond to which response.

Examples

For example, specifying a response_level of 52.3 followed with compute probabilities will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```plaintext
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the num_response_levels key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).
The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                        6.0e+04 6.5e+04 7.0e+04
                        3.5e+05 4.0e+05 4.5e+05
variables,
  normal_uncertain = 2
    means    = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = ‘TF1n’ ‘TF2n’
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = ‘TF1u’ ‘TF2u’
  weibull_uncertain = 2
    alphas    = 12., 30.
    betas     = 250., 590.
    descriptors = ‘TF1w’ ‘TF2w’
  histogram_bin_uncertain = 2
    num_pairs = 3 4
    abscissas = 5 8 10 .1 .2 .3 .4
    counts   = 17 21 0 12 24 12 0
    descriptors = ‘TF1h’ ‘TF2h’
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

```
Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
</tbody>
</table>
```
6.2. METHOD

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
<tr>
<td>3.5000000000e+05</td>
<td>4.0000000000e+05</td>
<td>8.6000000000e-06</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>4.5000000000e+05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e+00</td>
<td>5.5000000000e+01</td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e+00</td>
<td>3.8000000000e+01</td>
<td></td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e+00</td>
<td>2.3000000000e+01</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e+00</td>
<td>6.1000000000e+01</td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e+00</td>
<td>2.9000000000e+01</td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e+00</td>
<td>9.0000000000e+01</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e+00</td>
<td>5.2000000000e+01</td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e+00</td>
<td>9.0000000000e+01</td>
<td></td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

num_response_levels

- Keywords Area
- method
- global_evidence
- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions
Description

The num_response_levels keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If num_response_levels is not specified, the response_levels will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```plaintext
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5

compute
```

• Keywords Area
• method
• global_evidence
• response_levels
• compute

Selection of statistics to compute at each response level

Specification

Alias: none
Argument(s): none
Default: probabilities

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword Description</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>probabilities</td>
<td>Computes probabilities associated with response levels</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>gen_reliabilities</th>
<th>Computes generalized reliabilities associated with response levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td></td>
<td>Compute system reliability (series or parallel)</td>
</tr>
</tbody>
</table>

Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

**Default Behavior**

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

**Expected Output**

The type of statistics specified by `compute` will be reported for each response level.

**Usage Tips**

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

**Examples**

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
    probabilities
```

**Keywords Area**

- `method`
- `global_evidence`
- `response_levels`
- `compute`
- `probabilities`

Computes probabilities associated with response levels
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
  Argument(s): none

Description
The **probabilities** keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

**Default Behavior**
If **response_levels** is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the **probabilities** keyword should be specified in conjunction with the **compute** keyword.

**Expected Outputs**
The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples
```
method sampling
  sample_type random
  samples = 100 seed = 1
compute probabilities
complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                 6.0e+04 6.5e+04 7.0e+04
                 3.5e+05 4.0e+05 4.5e+05
```

Computes generalized reliabilities associated with response levels

Specification
Alias: none
  Argument(s): none
6.2. METHOD

Description

The gen_reliabilities keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If response_levels is specified, the generalized reliabilities are not computed by default. To change this behavior, the gen_reliabilities keyword should be specified in conjunction with the compute keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100  seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                       6.0e+04 6.5e+04 7.0e+04
                       3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

```
system
```

• Keywords Area

• method

• global_evidence

• response_levels

• compute

• system

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aggregate response statistics assuming a series system</td>
<td>series</td>
<td>system</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| parallel | Aggregate response statistics assuming a parallel system |

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

```
series
  - Keywords Area
  - method
  - global_evidence
  - response_levels
  - compute
  - system
  - series
```

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

```
parallel
  - Keywords Area
  - method
  - global_evidence
  - response_levels
  - compute
  - system
```
6.2. **METHOD**

- **parallel**

  Aggregate response statistics assuming a parallel system

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

See parent keyword *system* for description.

**distribution**

- **Keywords Area**
- **method**
- **global, evidence**
- **distribution**

  Selection of cumulative or complementary cumulative functions

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cumulative</strong></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

**Default Behavior**

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

cumulative

- **Keywords Area**
- **method**
- **global_evidence**
- **distribution**
- **cumulative**

Computes statistics according to cumulative functions

Specification

**Alias:** none

**Argument(s):** none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.
6.2. **METHOD**

**Examples**

```
method sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```

**complementary**

- **Keywords Area**
- **method**
- **global_evidence**
- **distribution**
- **complementary**

Computes statistics according to complementary cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the **complementary** keyword must be appear in conjunction with the **distribution** keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

**Examples**

```
method sampling
  sample_type lhs
  samples = 10
  distribution complementary
```

**probability_levels**

- **Keywords Area**
- **method**
- **global_evidence**
- **probability_levels**

Specify probability levels at which to estimate the corresponding response value
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>num_probability_</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>-levels</td>
<td>Specify which</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>probability_-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>correspond to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>which response</td>
</tr>
</tbody>
</table>

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If probability_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```plaintext
method,
  sampling,
    samples = 100 seed = 1
    complementary distribution
    probability_levels = 1. 0.66 .33 0.
                        1. 0.8 .5 0.
                        1. 0.3 .2 0.
variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
    num_pairs = 3 4
    abscissas = 5 8 10 .1 .2 .3 .4
    counts = 17 21 0 12 24 12 0
    descriptors = 'TF1h' 'TF2h'
```


6.2. METHOD

histogram_point_uncertain
real = 1
num_pairs = 2
abscessas = 3 4
counts = 1 1
descriptors = 'TF3h'

interface,
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses,
response_functions = 3
no_gradients
no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower  Bin Upper  Density Value
---------  ---------  ------------
2.7604749078e+11  3.4221494996e+11  5.13874797e-12
3.4221494996e+11  4.0634975300e+11  5.14512311e-12
4.0634975300e+11  5.4196114379e+11  2.4334239039e-12

PDF for response_fn_2:

Bin Lower  Bin Upper  Density Value
---------  ---------  ------------
4.643154744e+04  5.6511827775e+04  1.9839945149e-05
5.6511827775e+04  6.1603813790e+04  5.891608390e-05
6.1603813790e+04  7.8702465755e+04  2.9242071306e-05

PDF for response_fn_3:

Bin Lower  Bin Upper  Density Value
---------  ---------  ------------
2.3796737090e+05  3.6997214153e+05  5.329425348e-06
3.6997214153e+05  3.8100966235e+05  9.060055634e-06
3.8100966235e+05  4.4111498127e+05  3.274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level  Probability Level  Reliability Index  General Rel Index
-----------------  -----------------  -----------------  -----------------  
2.7604749078e+11  1.0000000000e+00
3.4221494996e+11  6.0000000000e-01
4.0634975300e+11  3.0000000000e-01
5.4196114379e+11  0.0000000000e+00

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level  Probability Level  Reliability Index  General Rel Index
-----------------  -----------------  -----------------  -----------------  
4.643154744e+04  1.0000000000e+00
5.6511827775e+04  8.0000000000e-01
6.1603813790e+04  5.0000000000e-01
7.8702465755e+04  0.0000000000e+00

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level  Probability Level  Reliability Index  General Rel Index
-----------------  -----------------  -----------------  -----------------  
2.3796737090e+05  1.0000000000e+00
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.4111498127e+05  0.0000000000e+00
CHAPTER 6. KEYWORDS AREA

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_probability_levels

- Keywords Area
- method
- global_evidence
- probability_levels
- num_probability_levels

Specify which probability_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGRALIST
Default: probability_levels evenly distributed among response functions

Description

See parent page

gen_reliability_levels

- Keywords Area
- method
- global_evidence
- gen_reliability_levels

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST
Default: No CDF/CCDF response levels to compute
### Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

### Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_gen_reliability_levels**

- Keywords Area
- method
- global_evidence
- gen_reliability_levels
- num_gen_reliability_levels

Specify which gen_reliability_levels correspond to which response

### Specification

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** gen_reliability_levels evenly distributed among response functions

### Description

See parent page
rng

- Keywords Area
- method
- global_evidence
- rng

Selection of a random number generator

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** Mersenne twister ( mt19937 )

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rnum2</td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
<tr>
<td>mt19937</td>
<td></td>
<td></td>
<td>Generates pseudo-random numbers using the Pecos package</td>
</tr>
</tbody>
</table>

**Description**

The `rng` keyword is used to indicate a choice of random number generator.  

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**

The default is recommended, as the Mersenne twister is a higher quality random number generator.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

```
mt19937
```

- Keywords Area
- method
6.2. METHOD

- global_evidence
- rng
- mt19937

Generates random numbers using the Mersenne twister

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

The **mt19937** keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on Wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

**Default Behavior**

**mt19937** is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the **rng** keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (**mt19937**) is recommended.

**Examples**

```dakota
method sampling
  sample_type lhs
  samples = 10
  seed = 98765
  rng mt19937
```

**rnum2**

- **Keywords Area**
- **method**
- **global_evidence**
- **rng**
- **rnum2**

Generates pseudo-random numbers using the Pecos package

**Specification**

**Alias:** none  
**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description

The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

**Default Behavior**

rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

**samples**

- **Keywords Area**
- **method**
- **global_evidence**
- **samples**

Number of samples for sampling-based methods

Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent

Description

The samples keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance-based_decomp, we recommend hundreds to thousands of samples. Note that for variance-based_decomp, the number of simulations performed will be N*(dim+2).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```
seed

- Keywords Area
- method
- global_evidence
- seed

Seed of the random number generator

**Specification**

**Alias:** none

- **Argument(s):** INTEGER
- **Default:** system-generated (non-repeatable)

**Description**

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**
If not specified, the seed is randomly generated.

**Expected Output**
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**model_pointer**

- Keywords Area
- method
- global_evidence
- model_pointer

Identifier for model block to be used by a method
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

```plaintext
environment
   tabular_graphics_data
   method_pointer = 'UQ'

method
   id_method = 'UQ'
   model_pointer = 'SURR'
   sampling,
      samples = 10
      seed = 98765 rng rnum2
      response_levels = 0.1 0.2 0.6
         0.1 0.2 0.6
         0.1 0.2 0.6
      sample_type lhs
      distribution cumulative

model
   id_model = 'SURR'
   surrogate global,
   dace_method_pointer = 'DACE'
   polynomial quadratic

method
   id_method = 'DACE'
   model_pointer = 'DACE_M'
   sampling sample_type lhs
      samples = 121 seed = 5034 rng rnum2

model
   id_model = 'DACE_M'
   single
```
6.2. METHOD

interface_pointer = ‘I1’

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = ‘x1’ ‘x2’

interface
id_interface = ‘I1’
system asynch evaluation_concurrency = 5
analysis_driver = ‘text_book’

responses
response_functions = 3
no_gradients
no_hessians

6.2.56  global_interval_est

- Keywords Area
- method
- global_interval_est

Interval analysis using global optimization methods

Topics
This keyword is related to the topics:

- uncertainty_quantification
- epistemic_uncertainty_quantification_methods
- interval_estimation

Specification

**Alias:** nond_global_interval_est
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>Group 1</td>
<td>sbo</td>
<td>Use the surrogate based optimization method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ego</td>
<td>Use the Efficient Global Optimization method</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ea</td>
<td>lhs</td>
<td>rng</td>
<td>max_iterations</td>
</tr>
<tr>
<td>Use an evolutionary algorithm</td>
<td>Uses Latin Hypercube Sampling (LHS) to sample variables</td>
<td>Selection of a random number generator</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td>Optional</td>
<td>samples</td>
<td>seed</td>
<td>model_pointer</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Description**

In the global approach to interval estimation, one uses either a global optimization method or a sampling method to assess the bounds of the responses.

`global_interval_est` allows the user to specify several approaches to calculate interval bounds on the output responses.

- **lhs** - note: this takes the minimum and maximum of the samples as the bounds
- **ego**
- **sbo**
6.2. METHOD

- ea

Additional Resources
Refer to variable_support for information on supported variable types.

See Also
These keywords may also be of interest:

- global_evidence
- local_evidence
- local_interval_est

sbo
- Keywords Area
- method
- global_interval_est
- sbo

Use the surrogate based optimization method

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>gaussian_process</td>
<td>Gaussian Process surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build-points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>
A surrogate-based optimization method will be used. The surrogate employed in sbo is a Gaussian process surrogate.

The main difference between ego and the sbo approach is the objective function being optimized. ego relies on an expected improvement function, while in sbo, the optimization proceeds using an evolutionary algorithm (coliny_ea) on the Gaussian process surrogate: it is a standard surrogate-based optimization. Also note that the sbo option can support optimization over discrete variables (the discrete variables are relaxed) while ego cannot. This is not the same as surrogate_based_global.

gaussian_process

- Keywords Area
- method
- global_interval_est
- sbo
- gaussian_process

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword surfpack</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/One</td>
<td></td>
<td>dakota</td>
<td>Select the built in Gaussian Process surrogate</td>
</tr>
</tbody>
</table>

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the surfpack keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.
6.2. METHOD

surfpack

- Keywords Area
- method
- global_interval_est
- sbo
- gaussian_process
- surfpack

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none
Argument(s): none

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack. Several user options are available:

1. Optimization methods:
   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See optimization_method.
   The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:
   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See trend.

3. Correlation Lengths:
   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.

4. Ill-conditioning
   One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.
   Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.
   The surfpack model handles ill-conditioning internally by default, but behavior can be modified using
5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

**Gradient Enhanced Kriging**

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here “inexpensive” means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

dakota

- Keywords Area
- method
- `global_interval_est`
- sbo
- `gaussian_process`
- dakota

Select the built in Gaussian Process surrogate

**Specification**

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

**use_derivatives**

- **Keywords Area**
- **method**
- **global_interval_est**
- **sbo**
- **use_derivatives**

Use derivative data to construct surrogate models

**Specification**

Alias: none

**Argument(s):** none

**Default:** use function values only

**Description**

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it’s use with Surfpack Gaussian process is not recommended.
import_build_points_file

- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file
**Argument(s):** STRING
**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>custom_annotated</td>
<td>Selects annotated tabular file format</td>
<td></td>
</tr>
<tr>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g., surrogate-based optimization, or implicitly, e.g., efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform
6.2. Method

Examples

```python
method
    polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
```

- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file
- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify `custom_annotated_header_eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.
Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id  interface  x1       x2       obj_fn  nln_ineq_con_1  nln_ineq_con_2
1  NO_ID        0.9  1.1      0.0002  0.26      0.76
2  NO_ID       0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3  NO_ID      0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

custom.annotated

- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file
- custom.annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format
6.2. METHOD

| Required/- | Description of | Dakota Keyword | Dakota Keyword Description |
| Optional   | Group          |               |                           |
| Optional   | header         | eval_id       | interface_id              |
| Optional   | enable header  | enable eval   | enable interface          |
| Optional   | row in custom-annotated tabular file | ID column in custom-annotated tabular file | ID column in custom-annotated tabular file |

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```
CHAPTER 6. KEYWORDS AREA

header

- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated

eval_id

- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no eval_id column

Description

See description of parent custom_annotated
6.2. METHOD

interface_id

- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent custom.annotated

**freeform**

- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format
CHAPTER 6. KEYWORDS AREA

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:
```
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:
```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

active_only
- Keywords Area
- method
- global_interval_est
- sbo
- import_build_points_file
- active_only

Import only active variables from tabular data file

Topics
This keyword is related to the topics:
- file_formats
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

export_approx_points_file

- Keywords Area
- method
- global_interval_est
- sbo
- export_approx_points_file

Output file for evaluations of a surrogate model

Specification

Alias: export_points_file
Argument(s): STRING
Default: no point export to a file

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

Description

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.
Usage Tips
Dakota exports tabular data in one of three formats:
- annotated (default)
- custom_annotated
- freeform

annotated
- Keywords Area
- method
- global_interval_est
- sbo
- export_approx_points_file
- annotated
Selects annotated tabular file format

Topics
This keyword is related to the topics:
- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips
- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
6.2. METHOD

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1  NO_ID  0.9  1.1  0.0002  0.26   0.76
2  NO_ID  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3  NO_ID  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...```

custom_annotated

- Keywords Area

- method

- global_interval_est

- sbo

- export_approx_points_file

- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format
**CHAPTER 6. KEYWORDS AREA**

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id   x1   x2   obj_fn nln_ineq_con_1  nln_ineq_con_2
1  0.9   1.1   0.0002   0.26   0.76
2 0.90009 1.1  0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1  0.0002003604863 0.2598380081 0.760045
...```
6.2. METHOD

header
• Keywords Area
• method
• global_interval_est
• sbo
• export_approx_points_file
• custom_annotated
• header

Enable header row in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no header

Description
See description of parent custom_annotated

eval_id
• Keywords Area
• method
• global_interval_est
• sbo
• export_approx_points_file
• custom_annotated
• eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no eval_id column

Description
See description of parent custom_annotated
interface_id

- Keywords Area
- method
- global_interval_est
- sbo
- export_approx_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none

Argument(s): none

Default: no interface_id column

**Description**

See description of parent custom.annotated

**freeform**

- Keywords Area
- method
- global_interval_est
- sbo
- export_approx_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none

Argument(s): none

Default: annotated format
Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9   1.1   0.0002  0.26   0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

ego

- Keywords Area
- method
- global_interval_est
- ego

Use the Efficient Global Optimization method

Specification
Alias: none
Argument(s): none
### Description

In the case of ego, the efficient global optimization (EGO) method is used to calculate bounds. By default, the Surfpack GP (Kriging) model is used, but the Dakota implementation may be selected instead. If `use_derivatives` is specified the GP model will be built using available derivative data (Surfpack GP only).

See efficient_global for more information.

### gaussian_process

- **Keywords Area**
- **method**
- **global_interval_est**
- **ego**
- **gaussian_process**

Gaussian Process surrogate model

### Specification

**Alias:** kriging

**Argument(s):** none

**Default:** Surfpack Gaussian process

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>gaussian_process</td>
<td>Gaussian Process surrogate model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>export_approx_points_file</td>
<td>Output file for evaluations of a surrogate model</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>surfpack</td>
<td>Use the Surfpack version of Gaussian Process surrogates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dakota</td>
<td>Select the built in Gaussian Process surrogate</td>
</tr>
</tbody>
</table>

**Description**

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the `surfpack` keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

`surfpack`

- **Keywords Area**
- **method**
- **global_interval_est**
- **ego**
- **gaussian_process**
- **surfpack**

Use the Surfpack version of Gaussian Process surrogates

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See `optimization_method`.

   The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.
2. Trend Function:
The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See trend.

3. Correlation Lengths:
Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.

4. Ill-conditioning
One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The surfpack model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).
The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging
Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

dakota

- Keywords Area
- method
- global_interval_est
6.2. METHOD

- ego
- gaussian_process
- dakota

Select the built in Gaussian Process surrogate

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the gaussian_process dakota model can be found in[58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a “nugget,” but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

**use_derivatives**

- **Keywords Area**
- method
- global_interval_est
- ego
- use_derivatives

Use derivative data to construct surrogate models

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** use function values only
**Description**

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

**import_build_points_file**

- **Keywords Area**
- **method**
- **global_interval_est**
- **ego**
- **import_build_points_file**

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file  
**Argument(s):** STRING  
**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group (Choose One)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>tabular format</td>
<td></td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.
Usage Tips
Dakota parses input files without regard to whitespace, but the import_build_points_file must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

Examples

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

annotated

- Keywords Area
- method
- global_interval_est
- ego
- import_build_points_file
- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

**Usage Tips**
To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`.

Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.

When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

**Examples**

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```python
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1   NO_ID  0.9  1.1 0.0002  0.26  0.76
2   NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3   NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
... 
```

**custom_annotated**

- **Keywords Area**
  - **method**
  - **global_interval_est**
  - **ego**
  - **import_build_points_file**
  - **custom_annotated**

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- **file_formats**

**Specification**

- **Alias**: none
  - **Argument(s)**: none
  - **Default**: annotated format
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id  x1  x2  obj_fn nln_ineq_con_1 nln_ineq_con_2
1   0.9  1.1  0.0002  0.26  0.76
2  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
3  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```
header
- Keywords Area
- method
- global_interval_est
- ego
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

Alias: none
Argument(s): none
Default: no header

**Description**

See description of parent *custom_annotated*

eval_id
- Keywords Area
- method
- global_interval_est
- ego
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

Alias: none
Argument(s): none
Default: no eval_id column

**Description**

See description of parent *custom_annotated*
interface_id

- Keywords Area
- method
- global_interval_est
- ego
- import_build_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none

- Argument(s): none
- Default: no interface_id column

**Description**

See description of parent custom_annotated

freeform

- Keywords Area
- method
- global_interval_est
- ego
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none

- Argument(s): none
- Default: annotated format
CHAPTER 6. KEYWORDS AREA

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform

Resulting tabular file:

  0.9       1.1       0.0002     0.26       0.76
  0.90009   1.1  0.0001996404857  0.2601620081  0.759955
  0.89991   1.1  0.0002003604863  0.2598380081  0.760045
  ...
```

active_only
- Keywords Area
- method
- global_interval_est
- ego
- import_build_points_file
- active_only
Import only active variables from tabular data file

Topics
This keyword is related to the topics:
- file_formats
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

**export_approx_points_file**

- Keywords Area
- method
- global_interval_est
- ego
- export_approx_points_file

Output file for evaluations of a surrogate model

Specification

Alias: export_points_file
Argument(s): STRING
Default: no point export to a file

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/One)</td>
<td></td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

Description

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.
Usage Tips
Dakota exports tabular data in one of three formats:

- annotated (default)
- custom_annotated
- freeform

annotated
- Keywords Area
- method
- global_interval_est
- ego
- export_approx_points_file
- annotated

Selects annotated tabular file format

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
6.2. METHOD

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
... 
```

**custom_annotated**

- **Keywords Area**
- **method**
- **global_interval_est**
- **ego**
- **export_approx_points_file**
- **custom_annotated**

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:

- **file_formats**

Specification

**Alias:** none  
**Argument(s):** none  
**Default:** annotated format

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
Optional eval_id

Enable evaluation ID column in custom-annotated tabular file

Optional interface_id

Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

header

• Keywords Area

• method

• global_interval_est
6.2. METHOD

- ego
- export_approx_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* no header

**Description**

See description of parent *custom_annotated*

**eval_id**

- Keywords Area
- method
- global_interval_est
- ego
- export_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* no eval_id column

**Description**

See description of parent *custom_annotated*
interface_id

- Keywords Area
- method
- global_interval_est
- ego
- export_approx_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none

*Argument(s):* none

*Default:* no interface_id column

**Description**

See description of parent `custom_annotated`

```
freeform
```

- Keywords Area
- method
- global_interval_est
- ego
- export_approx_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none

*Argument(s):* none

*Default:* annotated format
6.2. METHOD

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

evaluation

- Keywords Area

- method

- global_interval_est

- ea

Use an evolutionary algorithm

Specification

Alias: none

Argument(s): none
Description

In this approach, the evolutionary algorithm from Coliny, coliny_ea, is used to perform the interval optimization with no surrogate model involved. Again, this option of ea can support interval optimization over discrete variables.

lhs

- Keywords Area
- method
- global_interval_est
- lhs

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none
Argument(s): none

Description

The lhs keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

By default, Latin Hypercube Sampling is used. To explicitly specify this in the Dakota input file, however, the lhs keyword must appear in conjunction with the sample_type keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

method
  sampling
    sample_type lhs
    samples = 20

rng

- Keywords Area
- method
- global_interval_est
- rng

Selection of a random number generator
6.2. METHOD

Specification

Alias: none

Argument(s): none

Default: Mersenne twister ( mt19937 )

<table>
<thead>
<tr>
<th>Required/-Optional Required</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose One</td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
</tbody>
</table>

| rng2 | Generates pseudo-random numbers using the Pecos package |

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rng2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rng2

mt19937

  - Keywords Area
  - method
  - global_interval_est
  - rng
  - mt19937

  Generates random numbers using the Mersenne twister
```

Specification

Alias: none

Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended.

Examples

```plaintext
method sampling
  sample_type lhs
  samples = 10
  seed = 98765
  rng mt19937
```

rnum2

- Keywords Area
- method
- global_interval_est
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none
Argument(s): none

Description

The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples

```plaintext
method sampling
  sample_type lhs
  samples = 10
  seed = 98765
  rng rnum2
```
6.2. METHOD

max_iterations

- Keywords Area
- method
- global_interval_est
- max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- global_interval_est
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL
Default: 1.e-4
Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration. Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_function_evaluations

- Keywords Area
- method
- global_interval_est
- max_function_evaluations

Stopping criteria based on number of function evaluations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none
Argument(s): INTEGER
Default: 1000

Description

The maximum number of function evaluations (default: 1000). See also max_iterations.
6.2. METHOD

samples

- Keywords Area
- method
- global_interval_est
- samples

Number of samples for sampling-based methods

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** method-dependent

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least \( \text{dim}+1 \) samples should be used, where “\( \text{dim} \)” is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least \( (\text{dim}+1)(\text{dim}+2)/2 \) samples are needed. For uncertainty quantification, we recommend at least \( 10\times \text{dim} \) samples. For variance-based decomps, we recommend hundreds to thousands of samples. Note that for variance-based decomps, the number of simulations performed will be \( N\times(\text{dim}+2) \).

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 20

seed

- Keywords Area
- method
- global_interval_est
- seed

Seed of the random number generator

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** system-generated (non-repeatable)
CHAPTER 6. KEYWORDS AREA

Description
The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples
```dakota
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

model_pointer
  * Keywords Area
  * method
  * global_interval_est
  * model_pointer

Identifier for model block to be used by a method
```

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.
Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.
See `block_pointer` for details about pointers.

Examples

```
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'Surr'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'Surr'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.57 bayes_calibration

  • Keywords Area
• method
• bayes_calibration

Bayesian calibration

**Topics**
This keyword is related to the topics:

• bayesian_calibration
• package_queso

**Specification**

**Alias:** nond_bayes_calibration  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required (Choose One)</strong></td>
<td>Group 1</td>
<td>queso</td>
<td>Markov Chain Monte Carlo algorithms from the QUESO package</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gpmsa</td>
<td>(Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Markov Chain Monte Carlo algorithm with Gaussian Process Surrogate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wasabi</td>
<td>(Experimental Method) Non-MCMC Bayesian inference using interval analysis</td>
</tr>
</tbody>
</table>
### Description

Bayesian calibration methods take prior information on parameter values (in the form of prior distributions) and observational data (e.g., from experiments) and produce posterior distributions on the parameter values. When the computational simulation is then executed with samples from the posterior parameter distributions, the results that are produced are consistent with (“agree with”) the experimental data. Calibrating parameters from a computational simulation model requires a ”likelihood function” that specifies the likelihood of observing a particular observation given the model and its associated parameterization; Dakota assumes a Gaussian likelihood function currently. The algorithms that produce the posterior distributions on model parameters are most commonly Monte Carlo Markov Chain (MCMC) sampling algorithms. MCMC methods require many samples, often tens of thousands, so in the case of model calibration, often emulators of the computational simulation are used. For more details on the algorithms underlying the methods, see the Dakota User’s manual.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dream</strong></td>
<td>DREAM (DiffeRential Evolution Adaptive Metropolis)</td>
</tr>
<tr>
<td><em>Optional</em></td>
<td><strong>standardized_space</strong></td>
</tr>
<tr>
<td></td>
<td>Perform Bayesian inference in standardized probability space</td>
</tr>
<tr>
<td><em>Optional</em></td>
<td><strong>calibrate_error_multipliers</strong></td>
</tr>
<tr>
<td></td>
<td>Calibrate hyper-parameter multipliers on the observation error covariance</td>
</tr>
<tr>
<td><em>Optional</em></td>
<td><strong>convergence_tolerance</strong></td>
</tr>
<tr>
<td></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td><em>Optional</em></td>
<td><strong>max_iterations</strong></td>
</tr>
<tr>
<td></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td><em>Optional</em></td>
<td><strong>samples</strong></td>
</tr>
<tr>
<td></td>
<td>Number of samples for sampling-based methods</td>
</tr>
<tr>
<td><em>Optional</em></td>
<td><strong>seed</strong></td>
</tr>
<tr>
<td></td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td><em>Optional</em></td>
<td><strong>model_pointer</strong></td>
</tr>
<tr>
<td></td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>
Dakota has three Bayesian calibration methods: QUESO, DREAM, and GPMSA, specified with `bayes_calibration qeso`, `bayes_calibration dream`, or `bayes_calibration gpmsa`, respectively. The QUESO method uses components from the QUESO library (Quantification of Uncertainty for Estimation, Simulation, and Optimization) developed at The University of Texas at Austin. Dakota uses its DRAM (Delayed Rejected Adaptive Metropolis) algorithm, and variants, for the MCMC sampling. DREAM (DiffeRential Evolution Adaptive Metropolis) is a method that runs multiple different chains simultaneously for global exploration, and automatically tunes the proposal covariance during the process by a self-adaptive randomized subspace sampling.[86]. GPMSA (Gaussian Process Models for Simulation Analysis) is an approach developed at Los Alamos National Laboratory. It constructs Gaussian process models to emulate the expensive computational simulation as well as model discrepancy. GPMSA also has extensive features for calibration, such as the capability to include a "model discrepancy" term and the capability to model functional data such as time series data.

The Bayesian capabilities are under active development. At this stage, the QUESO methods in Dakota are the most advanced and robust, followed by DREAM, followed by GPMSA, which is in prototype form at this time. Dakota also has an experimental WASABI capability for non-MCMC Bayesian inference; it is not yet ready for production use. Note that as of Dakota 6.2, the field responses and associated field data may be used with QUESO and DREAM. That is, the user can specify field simulation data and field experiment data, and Dakota will interpolate and provide the proper residuals to the Bayesian calibration.

**queso**

- Keywords Area
- method
- bayes_calibration
- qeso

Markov Chain Monte Carlo algorithms from the QUESO package

**Topics**

This keyword is related to the topics:

- bayesian_calibration
- package_qeso

**Specification**

<table>
<thead>
<tr>
<th>Argument(s): none</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>emulator</td>
<td>Use an emulator or surrogate model to evaluate the likelihood function</td>
</tr>
</tbody>
</table>
### Method

| Optional | **logit_transform** | Utilize the logit transformation to reduce sample rejection for bounded domains |
| Optional | **export_chain_points_file** | Export the MCMC chain to the specified filename |
| Optional (Choose One) | **dram** | Use the DRAM MCMC algorithm |
| **MCMC algorithm type (Group 1)** | **delayed_rejection** | Use the Delayed Rejection MCMC algorithm |
| | **adaptive_metropolis** | Use the Adaptive Metropolis MCMC algorithm |
| | **metropolis_hastings** | Use the Metropolis--Hastings MCMC algorithm |
| | **multilevel** | Use the multilevel MCMC algorithm |
| Optional | **rng** | Selection of a random number generator |
| Optional | **pre_solve** | Perform deterministic optimization for MAP before Bayesian calibration |
CHAPTER 6. KEYWORDS AREA

Optional | proposal-covariance | Defines the technique used to generate the MCMC proposal covariance.

**Description**

For the QUESO method, one can use an emulator in the MCMC sampling. This will greatly improve the speed, since the Monte Carlo Markov Chain will generate thousands of samples on the emulator instead of the real simulation code. However, in the case of fast running evaluations, we recommend the use of no emulator. An emulator may be specified with the keyword `emulator`, followed by a `gaussian_process` emulator, a `pce` emulator (polynomial chaos expansion), or a `sc` emulator (stochastic collocation). For the `gaussian_process` emulator, the user must specify whether to use the `surfpack` or `dakota` version of the Gaussian process. The user can define the number of samples `emulator_samples` from which the emulator should be built. It is also possible to build the Gaussian process from points read in from the `import_points_file` and to export approximation-based sample evaluations using `export_points_file`. For `pce` or `sc`, the user can define a `sparse_grid_level`.

In terms of the MCMC sampling, one can specify one of the following MCMC algorithms to use: `dram` for D-RAM (Delayed Rejection Adaptive Metropolis), `delayed_rejection` for delayed rejection only, `adaptive_metropolis` for adaptive metropolis only, `metropolis_hastings` for Metropolis Hastings, and `multilevel` for the multilevel MCMC algorithm.

There are a variety of ways the user can specify the proposal covariance matrix which is very important in governing the samples generated in the chain. The proposal covariance specifies the covariance structure of a multivariate normal distribution. The user can specify `proposal_covariance`, followed by `derivatives`, `prior`, `values`, or `filename`. The derivative specification involves forming the Hessian of the misfit function (the negative log likelihood). When derivative information is available inexpensively (e.g. from an emulator), the derived-based proposal covariance forms a more accurate proposal distribution, resulting in lower rejection rates and faster chain mixing. The prior setting involves constructing the proposal from the variance of the prior distributions of the parameters being calibrated. When specifying the proposal covariance with values or from a file, the user can choose to specify only the diagonals of the covariance matrix with `diagonal` or to specify the full covariance matrix with `matrix`.

There are two other controls for QUESO. The `pre_solve` option enables the user to start the chain at an optimal point, the Maximum A Posteriori (MAP) point. This is the point in parameter space that maximizes the log posterior, (defined as the log-likelihood minus the log_prior). A deterministic optimization method is used to obtain the MAP point, and the MCMC chain is then started at the best point found in the optimization. The second factor is a `logit_transform`, which performs an internal variable transformation from bounded domains to unbounded domains in order to reduce sample rejection due to an out-of-bounds condition.

Note that as of Dakota 6.2, the field data capability may be used with QUESO. That is, the user can specify field simulation data and field experiment data, and Dakota will interpolate and provide the proper residuals to the Bayesian calibration.

**emulator**

- Keywords Area
- method
- bayes_calibration
Use an emulator or surrogate model to evaluate the likelihood function

### Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/ Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>emulator type (Group 1)</td>
<td>gaussian_process</td>
<td>Gaussian Process surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>pce</td>
<td>Polynomial Chaos Expansion surrogate model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sc</td>
<td>Stochastic Collocation polynomial surrogate model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
</tbody>
</table>

**Description**

This keyword describes the type of emulator used when calculating the likelihood function for the Bayesian calibration. The emulator can be a Gaussian process, polynomial chaos expansion, or stochastic collocation.

- **gaussian_process**
  - Keywords Area
  - method
  - bayes_calibration
  - queso
  - emulator
  - gaussian_process

Gaussian Process surrogate model

**Specification**

**Alias:** kriging  
**Argument(s):** none  
**Default:** Surfpack Gaussian process
<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group 1</td>
<td>surfpack</td>
<td>Use the Surfpack version of Gaussian Process surrogates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dakota</td>
<td>Select the built in Gaussian Process surrogate</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>emulator_samples</td>
<td>Number of data points used to train the surrogate model or emulator</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>posterior_adaptive</td>
<td>Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>

**Description**

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the **surfpack** keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

**surfpack**

- Keywords Area
- method
- bayes_calibration
- qeso
- emulator
- gaussian_process
- surfpack

Use the Surfpack version of Gaussian Process surrogates

**Specification**

Alias: none

Argument(s): none
6.2. METHOD

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See optimization_method.

   The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See trend.

3. Correlation Lengths:

   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.

4. Ill-conditioning

   One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

   Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

   The surfpack model handles ill-conditioning internally by default, but behavior can be modified using 5. Gradient Enhanced Kriging (GEK).

   The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

   See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient
information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

dakota

- Keywords Area
- method
- bayes_calibration
- quso
- emulator
- gaussian_process
- dakota

Select the built in Gaussian Process surrogate

**Specification**

Alias: none

Argument(s): none

**Description**

A second version of GP surrogates was available in prior versions of Dakota. For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the gaussian_process dakota model can be found in[58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a “nugget,” but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.
6.2. METHOD

emulator_samples

- Keywords Area
- method
- bayes_calibration
- qeso
- emulator
- gaussian_process
- emulator_samples

Number of data points used to train the surrogate model or emulator

Specification

Alias: none
Argument(s): INTEGER

Description

This keyword refers to the number of build points or training points used to construct a Gaussian process emulator. If the user specifies a number of emulator_samples that is less than the minimum number of points required to build the GP surrogate, Dakota will augment the samples to obtain the minimum required.

posterior_adaptive

- Keywords Area
- method
- bayes_calibration
- qeso
- emulator
- gaussian_process
- posterior_adaptive

Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

Specification

Alias: none
Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description
Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples
bayes_calibration queso
   samples = 2000 seed = 348
delayed_rejection
euler
gaussian_process surfpack emulator_samples = 30
posterior_adaptive max_iterations = 10
proposal_covariance derivatives

import_build_points_file

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- gaussian_process
- import_build_points_file

File containing points you wish to use to build a surrogate

Specification
Alias: import_points_file
Argument(s): STRING
Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>feature</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freeform file format</td>
<td>optional</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>active_only</td>
<td>active</td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom annotated
- freeform

**Examples**

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- gaussian_process
- import_build_points_file
- annotated
```

Selects annotated tabular file format
CHAPTER 6.  KEYWORDS AREA

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

  Argument(s): none

  Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated_header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1       x2       obj_fn   nln_ineq_con_1   nln_ineq_con_2
1   NO_ID                0.9       1.1       0.0002   0.26              0.76
2   NO_ID          0.90009   1.10009   0.0001996404857  0.2601620081   0.759955
3   NO_ID          0.89991   1.10009   0.0002003604863  0.2598380081   0.760045
...
```
6.2. METHOD

**custom_annotated**

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- gaussian_process
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description
A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval id column, and interface id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior
The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id  x1   x2   obj_fn  nln_ineq_con_1  nln_ineq_con_2
1.0  0.9    1.1   0.0002  0.26          0.76
2.0  0.9009 1.1   0.0001996404857 0.2601620081 0.759955
3.0  0.899911.1  0.0002003604863 0.2598380081 0.760045
...```

header
- Keywords Area
- method
- bayes_calibration
- qeso
- emulator
- gaussian_process
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file
6.2. METHOD

Specification

Alias: none
  Argument(s): none
  Default: no header

Description

See description of parent custom_annotated

  eval_id
  • Keywords Area
  • method
  • bayes_calibration
  • queso
  • emulator
  • gaussian_process
  • import_build_points_file
  • custom_annotated
  • eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none
  Argument(s): none
  Default: no eval_id column

Description

See description of parent custom_annotated

  interface_id
  • Keywords Area
  • method
  • bayes_calibration
  • queso
  • emulator
  • gaussian_process
• `import_build_points_file`

• `custom.annotated`

• `interface.id`

Enable interface ID column in custom-annotated tabular file

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* no interface_id column

**Description**

See description of parent `custom.annotated`

**freeform**

• `Keywords.Area`

• `method`

• `bayes.calibration`

• `queso`

• `emulator`

• `gaussian.process`

• `import_build_points_file`

• `freeform`

Selects freeform file format

**Topics**

This keyword is related to the topics:

• `file_formats`

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* annotated format
6.2. METHOD

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
    tabular_data
        tabular_data_file = 'dakota_summary.dat'
        freeform
```

Resulting tabular file:

```
0.9   1.1   0.0002  0.26   0.76
0.90009  1.1  0.0001996404857  0.2601620081  0.759955
0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

active_only

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- gaussian_process
- import_build_points_file
- active_only

Import only active variables from tabular data file
Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none

Description
By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

pce

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- pce

Polynomial Chaos Expansion surrogate model

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> (Choose One)</td>
<td>Group 1</td>
<td>sparse_grid_level</td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
</tbody>
</table>
### Description

Selects a polynomial chaos expansion (PCE) surrogate model to use in the Bayesian likelihood calculations. When using PCE as a surrogate within the Bayesian framework, the PCE coefficients can be computed either from integration using a sparse grid or from regression using a random/unstructured data set.

### See Also

These keywords may also be of interest:

- `polynomial_chaos`

- `sparse.grid.level`

- `Keywords Area`

- `method`

- `bayes_calibration`

- `queso`

- `emulator`

- `pce`

- `sparse.grid.level`

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation.

### Specification

**Alias:** none

**Argument(s):** INTEGERLIST
Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with \texttt{sparse\_grid\_level} and, optionally, \texttt{dimension\_preference}). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non_nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the \texttt{dimension\_preference} specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full \texttt{sparse\_grid\_level} and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for quadrature order, the \texttt{sparse\_grid\_level} specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.

\begin{verbatim}
expansion_order
\end{verbatim}

- Keywords Area
- method
- bayes\_calibration
- queso
- emulator
- pce
- expansion_order

The (initial) order of a polynomial expansion

Specification

Alias: none

Argument(s): INTEGERLIST
### Description

When the `expansion_order` for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with `expansion_samples`). In this case, the expansion order \( p \) cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify \( p \) for a total-order expansion.

Linear regression (specified with either `collocation_points` or `collocation_ratio`). A total-order expansion is used and must be specified using `expansion_order` as described in the previous option. To avoid requiring the user to calculate \( N \) from \( n \) and \( p \), the `collocation_ratio` allows for specification of a constant factor applied to \( N \) (e.g., `collocation_ratio = 2.` produces samples = \( 2N \)). In addition, the default linear relationship with \( N \) can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes \( cN^o \) where \( c \) is the `collocation_ratio` and \( o \) is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes \( \frac{N^o}{n+1} \)). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the `tensor_grid` keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If `collocation_points` or `collocation_ratio` is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved.

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>collocation_ratio</td>
<td>Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>posterior_adaptive</td>
<td>Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>
using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the noise_tolerance, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders (1,...,expansion_order) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

**collocation_ratio**

- Keywords Area
- method
- bayes_calibration
- quo
- emulator
- pce
- expansion_order
- collocation_ratio

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

**Specification**

**Alias:** none

**Argument(s):** REAL

**Description**

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p, the collocation_ratio allows for specification of a constant factor applied to N (e.g., collocation_ratio = 2 produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes \( cN^o \) where \( c \) is the collocation_ratio and \( o \) is the ratio_order. The use_derivatives flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes \( \frac{cN^o}{n+1} \)).
6.2. METHOD

- **posterior_adaptive**
  - Keywords Area
  - method
  - bayes_calibration
  - queso
  - emulator
  - pce
  - expansion_order
  - posterior_adaptive

Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

**Specification**

Alias: none

- **Argument(s):** none

**Description**

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

**Examples**

```
bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator
  gaussian_process surfpack emulator_samples = 30
  posterior_adaptive max_iterations = 10
  proposal_covariance derivatives
```

**import_build_points_file**

- Keywords Area
  - method
  - bayes_calibration
  - queso
  - emulator
  - pce
  - expansion_order
  - import_build_points_file

File containing points you wish to use to build a surrogate
CHAPTER 6. KEYWORDS AREA

**Specification**

**Alias:** import_points_file

**Argument(s):** STRING

**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional(Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

| Optional | active_only | Import only active variables from tabular data file |

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**Examples**

```
method
  polynomial_choas
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
  
annotated
    - Keywords Area
    - method
```
6.2. METHOD

- bayes_calibration
- queso
- emulator
- pce
- expansion_order
- import_build_points_file
- annotated

Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* annotated format

**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

**Usage Tips**

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
CHAPTER 6. KEYWORDS AREA

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
evironment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated

Resulting tabular file:

%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

custom_annotated

- Keywords Area
- method
- bayes_calibration
- quo
- emulator
- pce
- expansion_order
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated format

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td></td>
<td>Description</td>
</tr>
</tbody>
</table>
### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = ’dakota_summary.dat’
custom_annotated header eval_id
```

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.00019964</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.25983800581</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
header

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- pce
- expansion_order
- import_build_points_file
- custom.annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent custom.annotated

eval.id

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- pce
- expansion_order
- import_build_points_file
- custom.annotated
- eval.id

Enable evaluation ID column in custom-annotated tabular file
6.2. METHOD

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent custom_annotated

interface_id

• Keywords Area
• method
• bayes_calibration
• qeso
• emulator
• pce
• expansion_order
• import_build_points_file
• custom_annotated
• interface_id

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent custom_annotated

freeform

• Keywords Area
• method
• bayes_calibration
• qeso
• emulator
• pce
• expansion_order
• import_build_points_file
• freeform

Selects freeform file format

Topics
This keyword is related to the topics:
• file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
• In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```dakota```

```dakota```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```
6.2. METHOD

active_only

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- pce
- expansion_order
- import_build_points_file
- active_only
  Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
  Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

collocation_points

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- pce
- collocation_points

Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.
### Specification

**Alias:** none  
**Argument(s):** INTEGERLIST

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>posterior_adaptive</td>
<td>Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>

### Description

Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.

- **posterior_adaptive**
  - Keywords Area
  - method
  - bayes_calibration
  - queso
  - emulator
  - pce
  - collocation_points
  - posterior_adaptive
    Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

### Specification

**Alias:** none  
**Argument(s):** none

### Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.
6.2. METHOD

Examples

```
bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack emulator_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives
```

import_build_points_file

- **Keywords Area**
- **method**
- **bayes_calibration**
- **queso**
- **emulator**
- **pce**
- **collocation_points**
- **import_build_points_file**

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file

**Argument(s):** STRING

**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional(Choose One)</td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td></td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom.annotated
- freeform

**Examples**

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated

- Keywords Area
- method
- bayes.calibration
- queso
- emulator
- pce
- collocation_points
- import_build_points_file
- annotated
```

Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

- file_formats
6.2. METHOD

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom.annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

environment
tabular_data
tabular_data_file = ‘dakota_summary.dat’
annotated

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

custom.annotated

- Keywords Area

- method

- bayes_calibration

- queso

- emulator
• pce
• collocation_points
• import_build_points_file
• custom_annotated

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:
• file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description
A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior
The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips
• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
6.2. METHOD

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```dakota
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id   x1   x2   obj_fn nln_ineq_con_1 nln_ineq_con_2
1   0.9   1.1   0.0002   0.26    0.76
2   0.90009 1.1   0.0001996404857   0.2601620081 0.759955
3   0.89991 1.1   0.0002003604863   0.2598380081 0.760045
...```

header

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- pce
- collocation_points
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated
CHAPTER 6. KEYWORDS AREA

eval_id

- Keywords Area
- method
- bayes_calibration
- qeso
- emulator
- pce
- collocation_points
- import_build_points_file
- custom.annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent `custom.annotated`

interface_id

- Keywords Area
- method
- bayes_calibration
- qeso
- emulator
- pce
- collocation_points
- import_build_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file
6.2. **METHOD**

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no interface_id column

**Description**

See description of parent `custom.annotated`

```
freeform
```

- **Keywords Area**
- method
- bayes.calibration
- queso
- emulator
- pce
- collocation.points
- import.build.points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** annotated format

**Description**

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

**Usage Tips**
• Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
    tabular_data
        tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

active_only

• Keywords Area

• method

• bayes_calibration

• queso

• emulator

• pce

• collocation_points

• import_build_points_file

• active_only

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

• file_formats
6.2. METHOD

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

```
sc
```

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- sc

Stochastic Collocation polynomial surrogate model

### Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td><code>sparse_grid_level</code></td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
</tbody>
</table>

Description

Selects stochastic collocation (SC) model to use in the Bayesian likelihood calculations. When using SC as a surrogate within the Bayesian framework, the build points (training points) for the stochastic collocation are constructed from a sparse grid.
See Also

These keywords may also be of interest:

- stoch_collocation

\texttt{sparse\_grid\_level}

- Keywords Area
- method
- bayes_calibration
- quezo
- emulator
- sc
- \texttt{sparse\_grid\_level}

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation.

**Specification**

*Alias:* none
*Argument(s):* INTEGERLIST

**Description**

Multi-dimensional integration by the Smolyak sparse grid method (specified with \texttt{sparse\_grid\_level} and, optionally, \texttt{dimension\_preference}). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit \texttt{non\_nested} specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the \texttt{dimension\_preference} specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full \texttt{sparse\_grid\_level} and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for \texttt{quadrature\_order}, the \texttt{sparse\_grid\_level} specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.
6.2. METHOD

use_derivatives

- Keywords Area
- method
- bayes_calibration
- queso
- emulator
- use_derivatives

Use derivative data to construct surrogate models

Specification

Alias: none
- Argument(s): none
- Default: use function values only

Description

The use_derivatives flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

logit_transform

- Keywords Area
- method
- bayes_calibration
- queso
- logit_transform

Utilize the logit transformation to reduce sample rejection for bounded domains

Specification

Alias: none
- Argument(s): none
Description
The logit transformation performs an internal variable transformation from bounded domains to unbounded domains in order to reduce sample rejection due to an out-of-bounds condition.

Default Behavior
This option is experimental at present, and is therefore defaulted off.

Usage Tips
This option can be helpful when regions of high posterior density exist in the corners of a multi-dimensional bounded domain. In these cases, it may be difficult to generate feasible samples from the proposal density, such that transformation to unbounded domains may greatly reduce sample rejection rates.

Examples
```
method, bayes_calibration queso
  samples = 2000 seed = 348
  dram
  logit_transform
```

export_chain_points_file

- **Keywords Area**
- **method**
- **bayes_calibration**
- **queso**
- **export_chain_points_file**

Export the MCMC chain to the specified filename

Specification

Alias: none
Argument(s): STRING
Default: chain export to default filename

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional(Choose One)</td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
</tbody>
</table>
Description

The filename to which the final MCMC posterior chain will be exported.

**Default Behavior**
No export to file.

**Expected Output**
A tabular data file will be produced in the specified format (annotated by default) containing samples from the posterior distribution.

**Usage Tips**

**Additional Discussion**

annotated

- **Keywords Area**
  - method
  - bayes_calibration
  - queso
  - export_chain_points_file
  - annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

**Usage Tips**
• To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id.

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
evironment
  tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated

%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

custom_annotated

• Keywords Area

• method

• bayes_calibration

• queso

• export_chain_points_file

• custom_annotated

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:

• file_formats

Specification
Alias: none
Argument(s): none
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id    x1    x2    obj_fn   nl_nln_ineq_con_1   nl_nln_ineq_con_2
1           0.9    1.1     0.0002    0.26               0.76
2           0.90009 1.1  0.0001996404857 0.2601620081 0.759955
3           0.89991 1.1  0.0002003604863 0.2598380081 0.760045
...```
header
• Keywords Area
• method
• bayes_calibration
• queso
• export_chain_points_file
• custom.annotated
• header

Enable header row in custom-annotated tabular file

Specification
Alias: none
Argument(s): none

Description
See description of parent custom.annotated

eval_id
• Keywords Area
• method
• bayes_calibration
• queso
• export_chain_points_file
• custom.annotated
• eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none

Description
See description of parent custom.annotated
6.2. METHOD

interface_id

- Keywords Area
- method
- bayes_calibration
- quezo
- export_chain_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent custom.annotated

freeform

- Keywords Area
- method
- bayes_calibration
- quezo
- export_chain_points_file
- freeform

Selects freeform file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated
Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```python
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
</tr>
<tr>
<td>0.9009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
</tr>
<tr>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
</tr>
</tbody>
</table>
```

... 

dram

- Keywords Area
- method
- bayes_calibration
- queso
- dram

Use the DRAM MCMC algorithm

Topics

This keyword is related to the topics:

- bayesian_calibration
6.2. METHOD

Specification

Alias: none

Argument(s): none

Default: dram

Description

The type of Markov Chain Monte Carlo used. This keyword specifies the use of DRAM, (Delayed Rejection Adaptive Metropolis)[39].

Default Behavior

Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

Usage Tips

If the user knows very little about the proposal covariance, using dram is a recommended strategy. The proposal covariance is adaptively updated, and the delayed rejection may help improve low acceptance rates.

Examples

```
method,
    bayes_calibration queso
dram
    samples = 10000 seed = 348
```

delayed_rejection

- Keywords Area
- method
- bayes_calibration
- queso
- delayed_rejection

Use the Delayed Rejection MCMC algorithm

Topics

This keyword is related to the topics:

- bayesian_calibration

Specification

Alias: none

Argument(s): none

Default: dram
Description

This keyword specifies the use of the Delayed Rejection algorithm in which there can be a delay in rejecting samples from the chain. That is, the "DR" part of DRAM is used but the "AM" part is not, rather a regular Metropolis-Hastings algorithm is used.

Default Behavior

Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

Usage Tips

If the user knows something about the proposal covariance or the proposal covariance is informed through derivative information, using delayed_rejection is preferred over dram: the proposal covariance is already being informed by derivative information and the adaptive metropolis is not necessary.

Examples

method,
  bayes_calibration queso
  delayed_rejection
  samples = 10000 seed = 348

See Also

These keywords may also be of interest:

- proposal_covariance

adaptive_metropolis

- Keywords Area
- method
- bayes_calibration
- queso
- adaptive_metropolis

Use the Adaptive Metropolis MCMC algorithm

Topics

This keyword is related to the topics:

- bayesian_calibration

Specification

Alias: none
Argument(s): none
Default: dram
6.2. METHOD

Description

This keyword specifies the use of the Adaptive Metropolis algorithm. That is, the "AM" part of DRAM is used but the "DR" part is not: specifying this keyword activates only the Adaptive Metropolis part of the MCMC algorithm, in which the covariance of the proposal density is updated adaptively.

Default Behavior

Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

Usage Tips

If the user knows very little about the proposal covariance, but doesn’t want to incur the cost of using full dram with both delayed rejection and adaptive metropolis, specifying only adaptive_metropolis offers a good strategy.

Examples

```plaintext
method,
    bayes_calibration queso
    adaptive_metropolis
    samples = 10000 seed = 348
```

metropolis_hastings

- Keywords Area
- method
- bayes_calibration
- queso
- metropolis_hastings

Use the Metropolis-Hastings MCMC algorithm

Topics

This keyword is related to the topics:

- bayesian_calibration

Specification

Alias: none
Argument(s): none
Default: dram

Description

This keyword specifies the use of a Metropolis-Hastings algorithm for the MCMC chain generation. This means there is no delayed rejection and no adaptive proposal covariance updating as in DRAM.

Default Behavior

Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.
**Usage Tips**

If the user wants to use Metropolis-Hastings, possibly as a comparison to the other methods which involve more chain adaptation, this is the MCMC type to use.

**Examples**

```plaintext
method,
    bayes_calibration queso
    metropolis_hastings
    samples = 10000 seed = 348
```

**multilevel**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **queso**
- **multilevel**

Use the multilevel MCMC algorithm.

**Topics**

This keyword is related to the topics:

- **bayesian_calibration**

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** dram

**Description**

Selects the multilevel algorithm described in [70].

**Default Behavior**

Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

**Usage Tips**

The multilevel algorithm is a more experimental algorithm than the other MCMC approaches mentioned above. It works well in cases where the prior can be "evolved" to a posterior in a structured way. Currently, the multilevel option is not in production form.

**Examples**

```plaintext
method,
    bayes_calibration queso
    multilevel
    samples = 10000 seed = 348
```
rng

- Keywords Area
- method
- bayes_calibration
- queso
- rng

Selection of a random number generator

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** Mersenne twister (mt19937)

<table>
<thead>
<tr>
<th>Required/-Optional Required(Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rnum2</td>
<td>Generate pseudo-random numbers using the Pecos package</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The `rng` keyword is used to indicate a choice of random number generator.

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**

The default is recommended, as the Mersenne twister is a higher quality random number generator.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```
mt19937

- Keywords Area
- method
- bayes_calibration
- queso
- rng
- mt19937

Generates random numbers using the Mersenne twister

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The mt19937 keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

**Default Behavior**

mt19937 is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the rng keyword.

**Usage Tips**

Use of the Mersenne twister random number generator (mt19937) is recommended.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

rnum2

- Keywords Area
- method
- bayes_calibration
- queso
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

The rnum2 keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

rnum2 is not used by default. To change this behavior, it must be specified in conjunction with the rng keyword.

Usage Tips

Use of the Mersenne twister random number generator (mt19937) is recommended over rnum2.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

pre_solve

- Keywords Area
- method
- bayes_calibration
- queso
- pre_solve

Perform deterministic optimization for MAP before Bayesian calibration

Specification

Alias: none
Argument(s): none
Default: nip pre-solve for emulators

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqp</td>
<td>Uses a sequential quadratic programming method for underlying optimization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Description

When specified, Dakota will perform a deterministic gradient-based optimization to maximize the log posterior (log-likelihood - log prior). The Markov Chain in Bayesian calibration will be started at the best point found in the optimization (the MAP point). Note that both optimization methods available (sqp and nip) require gradients. The gradients will be assessed based on the model the user specifies (e.g. if an emulator is used, the gradients will be taken with respect to the emulator, otherwise they will be based on the user specification for the model, either finite-difference or analytic gradients.)

Default Behavior  No MAP pre-solve; simply start the MCMC process at the user-specified initial value.

Expected Output  When pre-solve is enabled, the output will include a deterministic optimization, followed by a Bayesian calibration. The final results will include the MAP point as well as posterior statistics from the MCMC chain.

Examples

```
method
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator
    pce sparse_grid_level = 2
  pre_solve nip

sqp
```

- Keywords Area
- method
- bayes_calibration
- queso
- pre_solve
- sqp

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none
Argument(s): none
6.2. METHOD

Description
Many uncertainty quantification methods solve a constrained optimization problem under the hood. The \texttt{sqp} keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

\texttt{nip}
- Keywords Area
- method
- \texttt{bayes\_calibration}
- \texttt{queso}
- \texttt{pre\_solve}
- \texttt{nip}

Uses a nonlinear interior point method for underlying optimization

Specification
Alias: none
Argument(s): none

Description
Many uncertainty quantification methods solve a constrained optimization problem under the hood. The \texttt{nip} keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

\texttt{proposal\_covariance}
- Keywords Area
- method
- \texttt{bayes\_calibration}
- \texttt{queso}
- \texttt{proposal\_covariance}

Defines the technique used to generate the MCMC proposal covariance.

Topics
This keyword is related to the topics:
- \texttt{bayesian\_calibration}

Specification
Alias: none
Argument(s): none
### Description

The proposal covariance is used to define a multivariate normal (MVN) jumping distribution used to create new points within a Markov chain. That is, a new point in the chain is determined by sampling within a MVN probability density with prescribed covariance that is centered at the current chain point. The accuracy of the proposal covariance has a significant effect on rejection rates and the efficiency of chain mixing.

#### Default Behavior

The default proposal covariance is prior when no emulator is present; derivatives when an emulator is present.

#### Expected Output

The effect of the proposal covariance is reflected in the MCMC chain values and the rejection rates, which can be seen in the diagnostic outputs from the QUESO solver within the QuesoDiagnostics directory.

#### Usage Tips

When derivative information is available inexpensively (e.g., from an emulator model), the derived-based proposal covariance forms a more accurate proposal distribution, resulting in lower rejection rates and faster chain mixing.

- **derivatives**
  - Keywords Area
  - method
  - bayes_calibration
6.2. METHOD

• queso
• proposal_covariance
• derivatives

Use derivatives to inform the MCMC proposal covariance.

Topics

This keyword is related to the topics:

• bayesian_calibration

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>proposal_updates</td>
<td>Restarts the MCMC chain with updated derivative-based proposal covariance.</td>
</tr>
</tbody>
</table>

Description

This keyword selection results in definition of the MCMC proposal covariance from the Hessian of the misfit function (negative log likelihood), where this Hessian is defined from either a Gauss-Newton approximation (using only first derivatives of the calibration terms) or a full Hessian (using values, first derivatives, and second derivatives of the calibration terms). If this Hessian is indeterminate, it will be corrected as described in[6]

Default Behavior  The default is prior based proposal covariance. This is a more advanced option that exploits structure in the form of the likelihood.

Expected Output

When derivatives are specified for defining the proposal covariance, the misfit Hessian and its inverse (the MVN proposal covariance) will be output to the standard output stream.

Usage Tips

The full Hessian of the misfit is used when either supported by the emulator in use (for PCE and surfpack GP, but not SC or dakota GP) or by the user’s response specification (Hessian type is not "no_hessians"), in the case of no emulator. When this full Hessian is indefinite and cannot be inverted to form the proposal covariance, fallback to the positive semi-definite Gauss-Newton Hessian is employed.

Since this proposal covariance is locally accurate, it should be updated periodically using the proposal_updates option. While the adaptive metropolis option can be used in combination with derivative-based preconditioning, it is generally preferable to instead increase the proposal update frequency due to the improved local accuracy of this approach.
Examples

method, bayes_calibration queso
    samples = 2000 seed = 348
delayed_rejection
emulator pce sparse_grid_level = 2
proposal_covariance derivatives # default proposal_updates

proposal_updates
• Keywords Area
• method
• bayes_calibration
• queso
• proposal_covariance
• derivatives
• proposal_updates

Restarts the MCMC chain with updated derivative-based proposal covariance.

Specification

Alias: none
Argument(s): INTEGER

Description

When employing derivative-based proposal covariance, this specification defines the number of restarts that are performed during the course of the total sample size of the MCMC chain. For each restart, a new chain is initiated from the final point in the previous acceptance chain using updated proposal covariance corresponding to the derivatives values at the new starting point.

Default Behavior

If proposal_updates is not specified, then the default frequency for restarting the chain with updated proposal covariance is every 100 samples.

Expected Output

Each restarted chain will generate a new QUESO header and sampling summary, and the chain diagnostics will be appended within the outputData directory.

Usage Tips

proposal_updates should be tailored to the size of the total chain, accounting for the relative expense of derivative-based proposal updates.

Examples

method, bayes_calibration queso
    samples = 2000 seed = 348
delayed_rejection
emulator pce sparse_grid_level = 2
proposal_covariance derivatives # default proposal_updates
    proposal_updates = 50 # restarted chains, each with 40 new points
6.2. METHOD

prior

• Keywords Area
• method
• bayes_calibration
• queso
• proposal_covariance
• prior

Uses the covariance of the prior distributions to define the MCMC proposal covariance.

Topics
This keyword is related to the topics:

• bayesian_calibration

Specification
Alias: none
Argument(s): none

Description
This keyword selection results in definition of the MCMC proposal covariance from the covariance of the prior distributions. This covariance is currently assumed to be diagonal without correlation.

Default Behavior
This is the default proposal_c covariance option.

Usage Tips
Since this proposal covariance is defined globally, the chain does not need to be periodically restarted using local updates to this proposal. However, it is usually effective to adapt the proposal using one of the adaptive metropolis MCMC options.

Examples
```
method,
    bayes_calibration queso
    samples = 2000 seed = 348
dram
    proposal_covariance prior
```

values

• Keywords Area
• method
• bayes_calibration
• queso
• proposal_covariance
• values

Specifies matrix values to use as the MCMC proposal covariance.

**Specification**

**Alias:** none

**Argument(s):** REALLIST

<table>
<thead>
<tr>
<th>Required/-Optional Required Group One</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>diagonal</td>
<td>Specifies the diagonal matrix format when specifying a user-specified proposal covariance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matrix</td>
<td>Specifies the full matrix format when specifying a user-specified proposal covariance.</td>
</tr>
</tbody>
</table>

**Description**

This keyword selection results in definition of the MCMC proposal covariance from user-specified matrix values. The matrix input format must be declared as either a full matrix or a matrix diagonal.

**Default Behavior**

This option is not the default, and generally implies special a priori knowledge from the user.

**Usage Tips**

This option is not supported for the case of transformations to standardized probability space.

**Examples**

```
method, bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    values ... # See leaf nodes for required format option
```

diagonal

• Keywords Area
• method
• bayes_calibration
6.2. METHOD

- queso
- proposal_covariance
- values
- diagonal

Specifies the diagonal matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

```
method,
bayes_calibration queso
samples = 1000 seed = 348
dram
proposal_covariance
diagonal values 1.0e6 1.0e-1
```

matrix

- Keywords Area
- method
- bayes_calibration
- queso
- proposal_covariance
- values
- matrix

Specifies the full matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none
**Description**

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

**Examples**

```plaintext
method,
    bayes_calibration queso
    samples = 1000 seed = 348
    dram
    proposal_covariance
        matrix values 1.0 0.1
                       0.1 2.0
```

**filename**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **queso**
- **proposal_covariance**
- **filename**

Uses a file to import a user-specified MCMC proposal covariance.

**Specification**

*Alias:* none

*Argument(s):* STRING

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diagonal</td>
<td></td>
<td>diagonal</td>
<td>Specifies the diagonal matrix format when importing a user-specified proposal covariance.</td>
</tr>
</tbody>
</table>
### Description
This keyword selection results in definition of the MCMC proposal covariance from importing data a user-specified filename. This import must be declared as either a full matrix or a matrix diagonal.

**Default Behavior**
This option is not the default, and generally implies special a priori knowledge from the user.

**Usage Tips**
This option is not supported for the case of transformations to standardized probability space.

### Examples
```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
dram
  proposal_covariance
    filename ... # See leaf nodes for required format option

  diagonal
```

### Specification
**Alias:** none
**Argument(s):** none

**Description**
When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.
Examples
method,
bayes_calibration queso
  samples = 1000 seed = 348
dram
  proposal_covariance
diagonal filename ‘dakota_cantilever_queso.diag.dat’

matrix
  Keywords Area
  method
  bayes_calibration
  queso
  proposal_covariance
  filename
  matrix
  Specifies the full matrix format when importing a user-specified proposal covariance.

Specification
Alias: none
Argument(s): none

Description
When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples
method,
bayes_calibration queso
  samples = 1000 seed = 348
dram
  proposal_covariance
 matrix filename ‘dakota_cantilever_queso.matrix.dat’

gpmsa
  Keywords Area
  method
  bayes_calibration
  gpmsa
  (Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Markov Chain Monte Carlo algorithm with Gaussian Process Surrogate
6.2. METHOD

Topics
This keyword is related to the topics:

- package_queso
- bayesian_calibration

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>emulator_samples</td>
<td></td>
<td>Number of data points used to train the surrogate model or emulator</td>
</tr>
<tr>
<td>Optional</td>
<td>import_build-points_file</td>
<td></td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>dram</td>
<td></td>
<td>Use the DRAM MCMC algorithm</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>delayed_rejection</td>
<td></td>
<td>Use the Delayed Rejection MCMC algorithm</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>adaptive_metropolis</td>
<td></td>
<td>Use the Adaptive Metropolis MCMC algorithm</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>metropolis_hastings</td>
<td></td>
<td>Use the Metropolis-Hastings MCMC algorithm</td>
</tr>
<tr>
<td>Optional</td>
<td>multilevel</td>
<td></td>
<td>Use the multilevel MCMC algorithm</td>
</tr>
<tr>
<td>Optional</td>
<td>rng</td>
<td></td>
<td>Selection of a random number generator</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

GPMSA (Gaussian Process Models for Simulation Analysis) is another approach that provides the capability for Bayesian calibration. The GPMSA implementation currently is an experimental capability and not ready for production use at this time. A key part of GPMSA is the construction of an emulator from simulation runs collected at various settings of input parameters. The emulator is a statistical model of the system response, and it is used to incorporate the observational data to improve system predictions and constrain or calibrate the unknown parameters. The GPMSA code draws heavily on the theory developed in the seminal Bayesian calibration paper by Kennedy and O’Hagan[55]. The particular approach in GPMSA has been developed by the Los Alamos group and document in[48]. GPMSA uses Gaussian process models in the emulation, but the emulator is actually a set of basis functions (e.g. from a singular value decomposition) which have GPs as the coefficients.

For the GPMSA method, one can define the number of samples which will be used in construction of the emulator, emulator_samples. The emulator involves Gaussian processes in GPMSA, so the user does not specify anything about emulator type. At this point, the only controls active for GPMSA are emulator_samples, seed and rng, and samples (the number of MCMC samples) and the type of MCMC algorithm (e.g. dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, or multilevel). NOTE: the GPMSA method is in a very preliminary, prototype state at this time. The user will need to modify certain data structures in the code for their particular application and recompile to run with GPMSA.

emulator_samples

- Keywords Area
- method
- bayes_calibration
- gpmsa
- emulator_samples

Number of data points used to train the surrogate model or emulator

Specification

Alias: none
Argument(s): INTEGER
6.2. METHOD

**Description**

This keyword refers to the number of build points or training points used to construct a Gaussian process emulator. If the user specifies a number of `emulator_samples` that is less than the minimum number of points required to build the GP surrogate, Dakota will augment the samples to obtain the minimum required.

**import_build_points_file**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **gpmsa**
- **import_build_points_file**

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file

**Argument(s):** STRING

**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td></td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annoted</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**
Dakota parses input files without regard to whitespace, but the import_build_points_file must be in one of three formats:

- annotated (default)
- custom.annotated
- freeform

**Examples**

```python
method
    polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none

Argument(s): none

Default: annotated format

**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

**Usage Tips**
6.2. METHOD

- To specify pre-Dakota 6.1 tabular format, which did not include interface id, specify custom_annotated header eval_id.

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.9009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
</tbody>
</table>

... custom_annotated

- Keywords Area
- method
- bayes_calibration
- gpmsa
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

**Description**

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

**Default Behavior**

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id     x1     x2     obj_fn  nln_ineq_con_1  nln_ineq_con_2
1  0.9     1.1    0.0002  0.26       0.76
2  0.90009 1.1   0.0001996404857 0.2601620081 0.759955
3  0.89991 1.1   0.0002003604863 0.2598380081 0.760045
...```
6.2. **METHOD**

- **header**
  - **Keywords Area**
  - **method**
  - **bayes_calibration**
  - **gpmsa**
  - **import_build_points_file**
  - **custom_annotated**
  - **header**

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no header

**Description**

See description of parent **custom_annotated**

- **eval_id**
  - **Keywords Area**
  - **method**
  - **bayes_calibration**
  - **gpmsa**
  - **import_build_points_file**
  - **custom_annotated**
  - **eval_id**

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no eval_id column

**Description**

See description of parent **custom_annotated**
interface_id

- Keywords Area
- method
- bayes_calibration
- gpmsa
- import_build_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* no interface_id column

**Description**

See description of parent custom_annotated

**freeform**

- Keywords Area
- method
- bayes_calibration
- gpmsa
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* annotated format
6.2. **METHOD**

**Description**

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The annotated format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows x num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

**Examples**

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
0.9  1.1  0.0002  0.26  0.76
0.90009  1.1  0.0001996404857  0.2601620081  0.759955
0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...
```

**Keywords Area**

- `method`
- `bayes_calibration`
- `import_build_points_file`

**Topics**

This keyword is related to the topics:

- `file_formats`
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword \texttt{active\_only} indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

\texttt{dram}

- Keywords Area
- method
- bayes\_calibration
- gpmsa
- dram

Use the DRAM MCMC algorithm

Topics

This keyword is related to the topics:

- bayesian\_calibration

Specification

Alias: none

Argument(s): none

Default: dram

Description

The type of Markov Chain Monte Carlo used. This keyword specifies the use of DRAM, (Delayed Rejection Adaptive Metropolis)[39].

\textbf{Default Behavior}

Five MCMC algorithm variants are supported: \texttt{dram}, \texttt{delayed\_rejection}, \texttt{adaptive\_metropolis}, \texttt{metropolis\_hastings}, and \texttt{multilevel}. The default is \texttt{dram}.

\textbf{Usage Tips}

If the user knows very little about the proposal covariance, using \texttt{dram} is a recommended strategy. The proposal covariance is adaptively updated, and the delayed rejection may help improve low acceptance rates.
6.2. METHOD

Examples

\begin{verbatim}
method, bayes_calibration queso
dram
samples = 10000 seed = 348
\end{verbatim}

delayed_rejection

- Keywords Area
- method
- bayes_calibration
- gpmsa
- delayed_rejection

Use the Delayed Rejection MCMC algorithm

Topics
This keyword is related to the topics:

- bayesian_calibration

Specification

Alias: none
Argument(s): none
Default: dram

Description
This keyword specifies the use of the Delayed Rejection algorithm in which there can be a delay in rejecting samples from the chain. That is, the "DR" part of DRAM is used but the "AM" part is not, rather a regular Metropolis-Hastings algorithm is used.

Default Behavior
Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

Usage Tips
If the user knows something about the proposal covariance or the proposal covariance is informed through derivative information, using delayed_rejection is preferred over dram: the proposal covariance is already being informed by derivative information and the adaptive metropolis is not necessary.

Examples

\begin{verbatim}
method, bayes_calibration queso
delayed_rejection
samples = 10000 seed = 348
\end{verbatim}
See Also
These keywords may also be of interest:

- proposal_covariance

adaptive_metropolis

- Keywords Area
- method
- bayes_calibration
- gpmsa
- adaptive_metropolis

Use the Adaptive Metropolis MCMC algorithm

Topics
This keyword is related to the topics:

- bayesian_calibration

Specification
Alias: none
Argument(s): none
Default: dram

Description
This keyword specifies the use of the Adaptive Metropolis algorithm. That is, the "AM" part of DRAM is used but the "DR" part is not: specifying this keyword activates only the Adaptive Metropolis part of the MCMC algorithm, in which the covariance of the proposal density is updated adaptively.

Default Behavior
Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

Usage Tips
If the user knows very little about the proposal covariance, but doesn’t want to incur the cost of using full dram with both delayed rejection and adaptive metropolis, specifying only adaptive_metropolis offers a good strategy.

Examples
method, bayes_calibration queso
adaptive_metropolis
samples = 10000 seed = 348
6.2. **METHOD**

**metropolis_hastings**

- **Keywords Area**
  - method
  - bayes_calibration
  - gpmsa
  - metropolis_hastings

Use the Metropolis-Hastings MCMC algorithm

**Topics**

This keyword is related to the topics:

- bayesian_calibration

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** dram

**Description**

This keyword specifies the use of a Metropolis-Hastings algorithm for the MCMC chain generation. This means there is no delayed rejection and no adaptive proposal covariance updating as in DRAM.

**Default Behavior**

Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

**Usage Tips**

If the user wants to use Metropolis-Hastings, possibly as a comparison to the other methods which involve more chain adaptation, this is the MCMC type to use.

**Examples**

```plaintext
method,
    bayes_calibration queso
metropolis_hastings
    samples = 10000 seed = 348
```

**multilevel**

- **Keywords Area**
  - method
  - bayes_calibration
  - gpmsa
  - multilevel

Use the multilevel MCMC algorithm.
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- bayesian_calibration

Specification
Alias: none

Argument(s): none

Default: dram

Description
Selects the multilevel algorithm described in[70].

Default Behavior
Five MCMC algorithm variants are supported: dram, delayed_rejection, adaptive_metropolis, metropolis_hastings, and multilevel. The default is dram.

Usage Tips
The multilevel algorithm is a more experimental algorithm than the other MCMC approaches mentioned above. It works well in cases where the prior can be "evolved" to a posterior in a structured way. Currently, the multilevel option is not in production form.

Examples
```
method, bayes_calibration queao
    multilevel
    samples = 10000 seed = 348
```

rng

- Keywords Area
- method
- bayes_calibration
- gpmsa
- rng

Selection of a random number generator

Specification
Alias: none

Argument(s): none

Default: Mersenne twister ( mt19937 )
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mt19937</td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rnum2</td>
<td>Generates pseudo-random numbers using the Pecos package</td>
</tr>
</tbody>
</table>

**Description**

The `rng` keyword is used to indicate a choice of random number generator.

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**

The default is recommended, as the Mersenne twister is a higher quality random number generator.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2

mt19937
  - Keywords Area
  - method
  - bayes_calibration
  - gpmsa
  - rng
  - mt19937

Generates random numbers using the Mersenne twister
```

**Specification**

**Alias:** none
**Argument(s):** none
Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: [http://en.wikipedia.org/wiki/Mersenne_twister](http://en.wikipedia.org/wiki/Mersenne_twister).

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937

rnum2

- Keywords Area
- method
- bayes_calibration
- gpmsa
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package
```

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.
6.2. METHOD

Examples

```method
sampling
  sample_type lhs
  samples = 10
  seed = 98765
  rng rnum2
```

pre_solve

- Keywords Area
- method
- bayes_calibration
- gpmsa
- pre_solve

Perform deterministic optimization for MAP before Bayesian calibration

Specification

Alias: none

Argument(s): none

Default: nip pre-solve for emulators

| Required/- | Description of Group 1 | Dakota Keyword | Dakota Keyword Description |
| Optional Required(Choose One) | | sqp | Uses a sequential quadratic programming method for underlying optimization |
| | | nip | Uses a nonlinear interior point method for underlying optimization |

Description

When specified, Dakota will perform a deterministic gradient-based optimization to maximize the log posterior (log-likelihood - log_prior). The Markov Chain in Bayesian calibration will be started at the best point found in the optimization (the MAP point). Note that both optimization methods available (sqp and nip) require gradients. The gradients will be assessed based on the model the user specifies (e.g. if an emulator is used, the gradients will be taken with respect to the emulator, otherwise they will be based on the user specification for the model, either finite-difference or analytic gradients.)

**Default Behavior** No MAP pre-solve; simply start the MCMC process at the user-specified initial value.
**Expected Output**  When pre-solve is enabled, the output will include a deterministic optimization, followed by a Bayesian calibration. The final results will include the MAP point as well as posterior statistics from the MCMC chain.

**Examples**

```plaintext
method
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator
    pce sparse_grid_level = 2
  pre_solve nip

sqp
- Keywords Area
- method
- bayes_calibration
- gpmsa
- pre_solve
- sqp

Uses a sequential quadratic programming method for underlying optimization
```

**Specification**

*Alias*: none

*Argument(s)*: none

**Description**

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `sqp` keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

```plaintext
nip
- Keywords Area
- method
- bayes_calibration
- gpmsa
- pre_solve
- nip

Uses a nonlinear interior point method for underlying optimization
```
6.2. METHOD

Specification

Alias: none
   Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

`proposal_covariance`

- Keywords Area
- method
- bayes_calibration
- gpmsa
- `proposal_covariance`

Defines the technique used to generate the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- `bayesian_calibration`

Specification

Alias: none
   Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required(<code>Choose One</code>)</td>
<td>Group 1</td>
<td>derivatives</td>
<td>Use derivatives to inform the MCMC proposal covariance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prior</td>
<td>Uses the covariance of the prior distributions to define the MCMC proposal covariance.</td>
</tr>
</tbody>
</table>
**Description**

The proposal covariance is used to define a multivariate normal (MVN) jumping distribution used to create new points within a Markov chain. That is, a new point in the chain is determined by sampling within a MVN probability density with prescribed covariance that is centered at the current chain point. The accuracy of the proposal covariance has a significant effect on rejection rates and the efficiency of chain mixing.

**Default Behavior**

The default proposal covariance is `prior` when no emulator is present; `derivatives` when an emulator is present.

**Expected Output**

The effect of the proposal covariance is reflected in the MCMC chain values and the rejection rates, which can be seen in the diagnostic outputs from the QUESO solver within the `QuesoDiagnostics` directory.

**Usage Tips**

When derivative information is available inexpensively (e.g., from an emulator model), the derived-based proposal covariance forms a more accurate proposal distribution, resulting in lower rejection rates and faster chain mixing.

**derivatives**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **gpmsa**
- **proposal_covariance**
- **derivatives**

Use derivatives to inform the MCMC proposal covariance.

**Topics**

This keyword is related to the topics:

- **bayesian_calibration**
6.2. METHOD

Specification

Alias: none
  Argument(s): none
## Description

This keyword selection results in definition of the MCMC proposal covariance from the Hessian of the misfit function (negative log likelihood), where this Hessian is defined from either a Gauss-Newton approximation (using only first derivatives of the calibration terms) or a full Hessian (using values, first derivatives, and second derivatives of the calibration terms). If this Hessian is indeterminate, it will be corrected as described in [6].

**Default Behavior**  The default is prior-based proposal covariance. This is a more advanced option that exploits structure in the form of the likelihood.

**Expected Output**  When derivatives are specified for defining the proposal covariance, the misfit Hessian and its inverse (the MVN proposal covariance) will be output to the standard output stream.

**Usage Tips**  

The full Hessian of the misfit is used when either supported by the emulator in use (for PCE and surfpack GP, but not SC or dakota GP) or by the user's response specification (Hessian type is not "no_hessians"), in the case of no emulator. When this full Hessian is indefinite and cannot be inverted to form the proposal covariance, fallback to the positive semi-definite Gauss-Newton Hessian is employed.

Since this proposal covariance is locally accurate, it should be updated periodically using the proposal_updates option. While the adaptive metropolis option can be used in combination with derivative-based preconditioning, it is generally preferable to instead increase the proposal update frequency due to the improved local accuracy of this approach.

## Examples

```plaintext
method, 
bayes_calibration queso 
samples = 2000 seed = 348 
delayed_rejection 
emulator pce sparse_grid_level = 2 
proposal_covariance derivatives # default proposal_updates
```

**propose_updates**  

- Keywords Area  
- method  
- bayes_calibration  
- gpmsa  
- proposal_covariance  
- derivatives
6.2. METHOD

- **proposal_updates**

  Restarts the MCMC chain with updated derivative-based proposal covariance.

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

When employing derivative-based proposal covariance, this specification defines the number of restarts that are performed during the course of the total sample size of the MCMC chain. For each restart, a new chain is initiated from the final point in the previous acceptance chain using updated proposal covariance corresponding to the derivatives values at the new starting point.

**Default Behavior**

If proposal_updates is not specified, then the default frequency for restarting the chain with updated proposal covariance is every 100 samples.

**Expected Output**

Each restarted chain will generate a new QUESO header and sampling summary, and the chain diagnostics will be appended within the outputData directory.

**Usage Tips**

proposal_updates should be tailored to the size of the total chain, accounting for the relative expense of derivative-based proposal updates.

**Examples**

```plaintext
method, bayes_calibration queso
  samples = 2000 seed = 348
delayed_rejection
  emulator pce sparse_grid_level = 2
  proposal_covariance derivatives
  proposal_updates = 50 # restarted chains, each with 40 new points

prior
- Keywords Area
- method
- bayes_calibration
- gpmsa
- proposal_covariance
- prior

Uses the covariance of the prior distributions to define the MCMC proposal covariance.

**Topics**

This keyword is related to the topics:

- bayesian_calibration
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none

Description

This keyword selection results in definition of the MCMC proposal covariance from the covariance of the prior distributions. This covariance is currently assumed to be diagonal without correlation.

Default Behavior
This is the default proposal covariance option.

Usage Tips
Since this proposal covariance is defined globally, the chain does not need to be periodically restarted using local updates to this proposal. However, it is usually effective to adapt the proposal using one of the adaptive metropolis MCMC options.

Examples

method,
bayes_calibration queso
samples = 2000 seed = 348
dram
proposal_covariance prior

values

- Keywords Area
- method
- bayes_calibration
- gpmsa
- proposal_covariance
- values

Specifies matrix values to use as the MCMC proposal covariance.

Specification

Alias: none
Argument(s): REALLIST
### Description

This keyword selection results in definition of the MCMC proposal covariance from user-specified matrix values. The matrix input format must be declared as either a full matrix or a matrix diagonal.

**Default Behavior**

This option is not the default, and generally implies special a priori knowledge from the user.

**Usage Tips**

This option is not supported for the case of transformations to standardized probability space.

### Examples

```plaintext
method,
bayes_calibration queso
samples = 1000 seed = 348
dram
proposal_covariance
  values ... # See leaf nodes for required format option
```

**diagonal**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **gpmsa**
- **proposal_covariance**
- **values**
- **diagonal**

 Specifies the diagonal matrix format when specifying a user-specified proposal covariance.

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group 1</td>
<td>diagonal</td>
<td>Description</td>
</tr>
<tr>
<td>Required(Choose One)</td>
<td></td>
<td>matrix</td>
<td>Specifies the full matrix format when specifying a user-specified proposal covariance.</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

method,
    bayes_calibration queso
    samples = 1000 seed = 348
    dram
    proposal_covariance
        diagonal values 1.0e6 1.0e-1

matrix

• Keywords Area
• method
• bayes_calibration
• gpmsa
• proposal_covariance
• values
• matrix

Specifies the full matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none
Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples

method,
    bayes_calibration queso
    samples = 1000 seed = 348
    dram
    proposal_covariance
        matrix values 1.0 0.1
        0.1 2.0
filename

- Keywords Area
- method
- bayes_calibration
- gpmsa
- proposal_covariance
- filename

Uses a file to import a user-specified MCMC proposal covariance.

**Specification**

**Alias:** none

**Argument(s):** STRING

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td></td>
<td>diagonal</td>
<td>Specifies the diagonal matrix format when importing a user-specified proposal covariance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matrix</td>
<td>Specifies the full matrix format when importing a user-specified proposal covariance.</td>
</tr>
</tbody>
</table>

**Description**

This keyword selection results in definition of the MCMC proposal covariance from importing data a user-specified filename. This import must be declared as either a full matrix or a matrix diagonal.

**Default Behavior**

This option is not the default, and generally implies special a priori knowledge from the user.

**Usage Tips**

This option is not supported for the case of transformations to standardized probability space.

**Examples**

```
method,
bayes_calibration queso
samples = 1000 seed = 348
dram
proposal_covariance
  filename ... # See leaf nodes for required format option
```
diagonal

• Keywords Area
• method
• bayes_calibration
• gpmsa
• proposal_covariance
• filename
• diagonal

Specifies the diagonal matrix format when importing a user-specified proposal covariance.

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

**Examples**

\[
\text{method, bayes\_calibration queso samples = 1000 seed = 348 dram}
\]
\[
\text{proposal\_covariance diagonal filename 'dakota\_cantilever\_queso diag.dat'}
\]

matrix

• Keywords Area
• method
• bayes_calibration
• gpmsa
• proposal_covariance
• filename
• matrix

Specifies the full matrix format when importing a user-specified proposal covariance.
6.2. **METHOD**

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

**Examples**

```plaintext
method,
    bayes_calibration queso
    samples = 1000 seed = 348
dram
    proposal_covariance
    matrix filename 'dakota_cantilever_queso.matrix.dat'
```

**wasabi**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **wasabi**

*(Experimental Method)* Non-MCMC Bayesian inference using interval analysis

**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optional</strong></td>
<td></td>
<td>emulator</td>
<td>Use an emulator or surrogate model to evaluate the likelihood function</td>
</tr>
<tr>
<td><strong>Required</strong></td>
<td></td>
<td>data_distribution</td>
<td><em>(Experimental Capability)</em> Specify the distribution of the experimental data</td>
</tr>
</tbody>
</table>
### Description

Offers an alternative to Markov Chain Monte Carlo-based Bayesian inference. This is a nascent capability, not yet ready for production use.

**Usage Guidelines:** The WASABI method requires an emulator model.

**Attention:** While the `emulator` specification for WASABI includes the keyword `posterior.adaptive`, it is not yet operational.

### Examples

```python
method
  bayes_calibration
    wasabi

emulator

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
```

Use an emulator or surrogate model to evaluate the likelihood function

### Specification

**Alias:** none

**Argument(s):** none
### Description

This keyword describes the type of emulator used when calculating the likelihood function for the Bayesian calibration. The emulator can be a Gaussian process, polynomial chaos expansion, or stochastic collocation.

- **gaussian_process**
  - Keywords Area
  - method
  - bayes_calibration
  - wasabi
  - emulator
  - gaussian_process

  Gaussian Process surrogate model

### Specification

**Alias:** kriging  
**Argument(s):** none  
**Default:** Surfpack Gaussian process
### Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the `surfpack` keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the Dakota version is deprecated and intended to be removed in a future release.**

**surfpack**
- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
- surfpack

Use the Surfpack version of Gaussian Process surrogates

### Specification

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:
   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See optimization_method.

   The total number of evaluations of the likelihood function can be controlled using the max_trials keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:
   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See trend.

3. Correlation Lengths:
   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.

4. Ill-conditioning
   One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

   Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

   The surfpack model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

   The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

   See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient
information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

dakota

• Keywords Area
• method
• bayes_calibration
• wasabi
• emulator
• gaussian_process
• dakota

Select the built in Gaussian Process surrogate

**Specification**

Alias: none

Argument(s): none

**Description**

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.** Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the gaussian_process dakota model can be found in[58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a “nugget,” but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.
### emulator_samples

- **Keywords Area**
- **method**
- **bayes_calibration**
- **wasabi**
- **emulator**
- **gaussian_process**
- **emulator_samples**

  Number of data points used to train the surrogate model or emulator

**Specification**

Alias: none

Argument(s): INTEGER

**Description**

This keyword refers to the number of build points or training points used to construct a Gaussian process emulator. If the user specifies a number of `emulator_samples` that is less than the minimum number of points required to build the GP surrogate, Dakota will augment the samples to obtain the minimum required.

### posterior_adaptive

- **Keywords Area**
- **method**
- **bayes_calibration**
- **wasabi**
- **emulator**
- **gaussian_process**
- **posterior_adaptive**

  Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

**Specification**

Alias: none

Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

bayes_calibration queso
samples = 2000 seed = 348
delayed_rejection
emulator
  gaussian_process surfpack emulator_samples = 30
  posterior_adaptive max_iterations = 10
  proposal_covariance derivatives

import_build_points_file

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
- import_build_points_file

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file
Argument(s): STRING
Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>freeform</th>
<th>Selects freeform file format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**Examples**

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

  annotated
    Keywords Area
    method
    bayes_calibration
    wasabi
    emulator
    gaussian_process
    import_build_points_file
    annotated
    Selects annotated tabular file format
```
Topics
This keyword is related to the topics:

- **file Formats**

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**
By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

**Usage Tips**

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify `custom_annotated header eval_id`

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```
custom_annotated

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
- import_build.points_file
- custom_annotated

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none

Argument(s): none

Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id  x1  x2  obj_fn  nln_ineq_con_1  nln_ineq_con_2  
1  0.9  1.1  0.0002  0.26  0.76  
2  0.90009  1.1  0.0001996404857  0.2601620081  0.759955  
3  0.89991  1.1  0.0002003604863  0.2598380081  0.760045  
...  
```

header

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file
6.2. METHOD

**Specification**

Alias: none

Argument(s): none

Default: no header

**Description**

See description of parent custom_annotated

`eval_id`

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

Alias: none

Argument(s): none

Default: no eval_id column

**Description**

See description of parent custom_annotated

`interface_id`

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
CHAPTER 6. KEYWORDS AREA

- import_build_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent `custom.annotated`

**freeform**

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format
6.2. METHOD

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

active_only

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- gaussian_process
- import_build_points_file
- active_only

Import only active variables from tabular data file
Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none

Description
By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword active_only indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

pce

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce

Polynomial Chaos Expansion surrogate model

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required(Choose One)</td>
<td>Group 1</td>
<td>sparse_grid_level</td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>expansion_order</th>
<th>The (initial) order of a polynomial expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>collocation_points</td>
<td>Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.</td>
</tr>
</tbody>
</table>

**Description**

Selects a polynomial chaos expansion (PCE) surrogate model to use in the Bayesian likelihood calculations. When using PCE as a surrogate within the Bayesian framework, the PCE coefficients can be computed either from integration using a sparse grid or from regression using a random/unstructured data set.

**See Also**

These keywords may also be of interest:

- polynomial_chaos
- sparse_grid_level
- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- sparse_grid_level

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation.

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST
CHAPTER 6. KEYWORDS AREA

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.

```
expansion_order
```

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order

The (initial) order of a polynomial expansion

Specification

Alias: none

Argument(s): INTEGERLIST
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>collocation_ratio</td>
<td>Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>posterior_adaptive</td>
<td>Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>

Description

When the expansion_order for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with expansion_samples). In this case, the expansion order $p$ cannot be inferred from the numerical integration specification and it is necessary to provide an expansion_order to specify $p$ for a total-order expansion.

Linear regression (specified with either collocation_points or collocation_ratio). A total-order expansion is used and must be specified using expansion_order as described in the previous option. To avoid requiring the user to calculate $N$ from $n$ and $p$, the collocation_ratio allows for specification of a constant factor applied to $N$ (e.g., collocation_ratio = 2. produces samples = 2N). In addition, the default linear relationship with $N$ can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes $cN^o$ where $c$ is the collocation_ratio and $o$ is the ratio_order. The use_derivatives flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{N_p}{n+1}$). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the tensor_grid keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If collocation_points or collocation_ratio is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved
using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the noise_tolerance, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders (1,...,expansion_order) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

**collocation_ratio**

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- collocation_ratio

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

**Specification**

**Alias:** none

**Argument(s):** REAL

**Description**

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p, the collocation_ratio allows for specification of a constant factor applied to N (e.g., collocation_ratio = 2. produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes $cN^o$ where $c$ is the collocation_ratio and $o$ is the ratio_order. The use_derivatives flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).
6.2. **METHOD**

**posterior_adaptive**
- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- posterior_adaptive

Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

**Examples**

```plaintext
bayes_calibration queso
   samples = 2000 seed = 348
   delayed_rejection
   emulator
      gaussian_process surfpack emulator_samples = 30
      posterior_adaptive max_iterations = 10
      proposal_covariance derivatives
```

**import_build_points_file**

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- import_build_points_file

File containing points you wish to use to build a surrogate
CHAPTER 6. KEYWORDS AREA

Specification

Alias: import_points_file
Argument(s): STRING
Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

| Optional | active_only | Import only active variables from tabular data file |

Description

The import_build_points_file allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior
Be default, methods do not import points from a file.

Usage Tips
Dakota parses input files without regard to whitespace, but the import_build_points_file must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

Examples

```
method polynomial_chaos
  expansion_order = 4
  import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
  - Keywords Area
  - method
```
6.2. METHOD

- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- import_build_points_file
- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

**Usage Tips**

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
CHAPTER 6. KEYWORDS AREA

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```python
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

custom_annotated

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
</table>
6.2. METHOD

| Optional | header | Enable header row in custom-annotated tabular file |
| Optional | eval_id | Enable evaluation ID column in custom-annotated tabular file |
| Optional | interface_id | Enable interface ID column in custom-annotated tabular file |

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id   x1    x2    obj_fn  nln_ineq_con_1 nln_ineq_con_2
1     0.90000  1.1   0.0002  0.2601620081  0.760045
2     0.90009  1.1   0.0001996404857 0.2601620081  0.759955
3     0.89991  1.1  0.0002003604863 0.2598380081  0.760045
...```
CHAPTER 6. KEYWORDS AREA

header

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated

eval_id

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file
6.2. METHOD

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent custom_annotated

interface_id

• Keywords Area
• method
• bayes_calibration
• wasabi
• emulator
• pce
• expansion_order
• import_build_points_file
• custom_annotated
• interface_id

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent custom_annotated

freeform

• Keywords Area
• method
• bayes_calibration
• wasabi
• emulator
CHAPTER 6. KEYWORDS AREA

- pce
- expansion_order
- import_build_points_file
- freeform

Selects freeform file format

Topics

This keyword is related to the topics:
- fileFormats

Specification

Alias: none
Argument(s): none
Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
  0.9  1.1  0.0002  0.26  0.76
  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
  ...
```
6.2. METHOD

active_only
- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- expansion_order
- import_build_points_file
- active_only
  Import only active variables from tabular data file

Topics
This keyword is related to the topics:
- file_formats

Specification
Aliase: none
Argument(s): none

Description
By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword active_only indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

collocation_points
- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- collocation_points
  Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.
Specification

Alias: none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>posterior_adaptive</td>
<td>Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>

Description

Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.

- posterior_adaptive
  - Keywords Area
  - method
  - bayes_calibration
  - wasabi
  - emulator
  - pce
  - collocation_points
  - posterior_adaptive

Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.
6.2. METHOD

Examples

bayes_calibration queso
samples = 2000 seed = 348
delayed_rejection
delayed_rejection
gaussian_process surfpack emulator_samples = 30
posterior_adaptive max_iterations = 10
proposal_covariance derivatives

import_build_points_file

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- collocation_points
- import_build_points_file

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file
Argument(s): STRING
Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td></td>
<td>custom.annotated</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>active_only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>
**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**Examples**

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated

  - Keywords Area
  - method
  - bayes_calibration
  - wasabi
  - emulator
  - pce
  - collocation_points
  - import_build_points_file
  - annotated

Selects annotated tabular file format
```

**Topics**

This keyword is related to the topics:

- file_formats
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

• To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

custom_annotated

• Keywords Area

• method

• bayes_calibration

• wasabi

• emulator
• pce
• collocation_points
• import_build_points_file
• custom_annotated

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:

• file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description
A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior
The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips
• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
6.2. METHOD

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

header

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- collocation_points
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated
eval_id

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- collocation_points
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent custom_annotated

interface_id

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- pce
- collocation_points
- import_build_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: no interface_id column

Description

See description of parent custom_annotated

freeform

• Keywords Area
• method
• bayes_calibration
• wasabi
• emulator
• pce
• collocation_points
• import_build_points_file
• freeform

Selects freeform file format

Topics

This keyword is related to the topics:

• file_formats

Specification

Alias: none
Argument(s): none
Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```bash
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

**active_only**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **wasabi**
- **emulator**
- **pce**
- **collocation_points**
- **import_build_points_file**
- **active_only**

Import only active variables from tabular data file

**Topics**

This keyword is related to the topics:

- **file_formats**
6.2. METHOD

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

```
sc
```

- **Keywords Area**
  - method
  - bayes_calibration
  - wasabi
  - emulator
  - sc

Stochastic Collocation polynomial surrogate model

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>sparse_grid_level</td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
</tbody>
</table>

**Description**

Selects stochastic collocation (SC) model to use in the Bayesian likelihood calculations. When using SC as a surrogate within the Bayesian framework, the build points (training points) for the stochastic collocation are constructed from a sparse grid.
See Also

These keywords may also be of interest:

- `stoch_collocation`

`sparse_grid_level`

- `Keywords Area`
- `method`
- `bayes_calibration`
- `wasabi`
- `emulator`
- `sc`
- `sparse_grid_level`

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation.

Specification

**Alias:** none

**Argument(s):** INTEGERLIST

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non_nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.
6.2. METHOD

use_derivatives

- Keywords Area
- method
- bayes_calibration
- wasabi
- emulator
- use_derivatives

Use derivative data to construct surrogate models

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** use function values only

**Description**

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

data_distribution

- Keywords Area
- method
- bayes_calibration
- wasabi
- data_distribution

(Experimental Capability) Specify the distribution of the experimental data

**Specification**

**Alias:** none  
**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required (Choose One)</th>
<th>Group 1</th>
<th>gaussian</th>
<th>(Experimental Capability) Gaussian error distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>obs_data_filename</td>
<td>(Experimental Capability) Filename from which to read experimental data</td>
</tr>
</tbody>
</table>

**gaussian**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **wasabi**
- **data_distribution**
- **gaussian**

(Experimental Capability) Gaussian error distribution

**Specification**

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>means</td>
<td>(Experimental Capability) Means of Gaussian error distribution</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>covariance</td>
<td>(Experimental Capability) Covariance of a Gaussian error distribution</td>
</tr>
</tbody>
</table>

**means**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **wasabi**
- **data_distribution**
6.2. METHOD

- gaussian
- means

(Experimental Capability) Means of Gaussian error distribution

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**covariance**

- Keywords Area
- method
- bayes_calibration
- wasabi
- data_distribution
- gaussian
- covariance

(Experimental Capability) Covariance of a Gaussian error distribution

**Specification**

**Alias:** none

**Argument(s):** REALLIST

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>diagonal</td>
<td>Diagonal error covariance</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>matrix</td>
<td>Symmetric positive definite error covariance</td>
</tr>
</tbody>
</table>

**diagonal**

- Keywords Area
- method
- bayes_calibration
- wasabi
• data_distribution
• gaussian
• covariance
• diagonal

(Experimental Capability) Diagonal error covariance

Specification
Alias: none
Argument(s): none

matrix
• Keywords Area
• method
• bayes_calibration
• wasabi
• data_distribution
• gaussian
• covariance
• matrix

(Experimental Capability) Symmetric positive definite error covariance

Specification
Alias: none
Argument(s): none

obs_data_filename
• Keywords Area
• method
• bayes_calibration
• wasabi
• data_distribution
• obs_data_filename

(Experimental Capability) Filename from which to read experimental data
6.2. METHOD

Specification
Alias: none
Argument(s): STRING

**posterior_density_export_filename**
- Keywords Area
- method
- bayes_calibration
- wasabi
- **posterior_density_export_filename**
  (Experimental Capability) Filename for the exported posterior density

Specification
Alias: none
Argument(s): STRING

**posterior_samples_export_filename**
- Keywords Area
- method
- bayes_calibration
- wasabi
- **posterior_samples_export_filename**
  (Experimental Capability) Filename for the exported posterior samples

Specification
Alias: none
Argument(s): STRING

**posterior_samples_import_filename**
- Keywords Area
- method
- bayes_calibration
- wasabi
- **posterior_samples_import_filename**
  (Experimental Capability) Filename for imported posterior samples
**Specification**

**Alias**: none

**Argument(s)**: STRING

---

**generate_posterior_samples**

- Keywords Area
- method
- bayes_calibration
- wasabi
- generate_posterior_samples

*(Experimental Capability)* Generate random samples from the posterior density

---

**Specification**

**Alias**: none

**Argument(s)**: none

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>evaluate_posterior_density</td>
<td><em>(Experimental Capability)</em> Evaluate the posterior density and output to the specified file</td>
</tr>
</tbody>
</table>

---

**evaluate_posterior_density**

- Keywords Area
- method
- bayes_calibration
- wasabi
- generate_posterior_samples
- evaluate_posterior_density

*(Experimental Capability)* Evaluate the posterior density and output to the specified file

---

**Specification**

**Alias**: none

**Argument(s)**: none
6.2. METHOD

**dream**

- **Keywords Area**
  - method
  - bayes_calibra
tion
  - dream

DREAM (DiffeRential Evolution Adaptive Metropolis)

**Topics**

This keyword is related to the topics:

- bayesian_calibration

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>chains</td>
<td>chains</td>
<td>Number of chains in DREAM</td>
</tr>
<tr>
<td>Optional</td>
<td>num_cr</td>
<td>num_cr</td>
<td>Number of candidate points for each crossover.</td>
</tr>
<tr>
<td>Optional</td>
<td>crossover_chain_pairs</td>
<td>crossover_chain_pairs</td>
<td>Number of chains used in crossover.</td>
</tr>
<tr>
<td>Optional</td>
<td>gr_threshold</td>
<td>gr_threshold</td>
<td>Convergence tolerance for the Gelman-Rubin statistic</td>
</tr>
<tr>
<td>Optional</td>
<td>jump_step</td>
<td>jump_step</td>
<td>Number of generations a long jump step is taken</td>
</tr>
<tr>
<td>Optional</td>
<td>emulator</td>
<td>emulator</td>
<td>Use an emulator or surrogate model to evaluate the likelihood function</td>
</tr>
</tbody>
</table>

**Description**

The DiffeRential Evolution Adaptive Metropolis algorithm is described in [86]. For the DREAM method, one can define the number of chains used with chains (minimum 3). The total number of generations per chain in DREAM is the number of samples (samples) divided by the number of chains (chains). The number of chains randomly selected to be used in the crossover each time a crossover occurs is crossover_chain_pairs. There is an extra adaptation during burn-in, in which DREAM estimates a distribution of crossover probabilities that favors large jumps over smaller ones in each of the chains. Normalization is required to ensure that all of
the input dimensions contribute equally. In this process, a discrete number of candidate points for each crossover value is generated. This parameter is `num_cr`. The `gr_threshold` is the convergence tolerance for the Gelman-Rubin statistic which will govern the convergence of the multiple chain process. The integer `jump_step` forces a long jump every `jump_step` generations. For more details about these parameters, see [86].

**Attention:** While the emulator specification for DREAM includes the keyword `posterior_adaptive`, it is not yet operational.

**chains**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **dream**
- **chains**

Number of chains in DREAM

**Topics**

This keyword is related to the topics:

- **bayesian_calibration**

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** 3

**Description**

Number of chains in DREAM

**num_cr**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **dream**
- **num_cr**

Number of candidate points for each crossover.

**Topics**

This keyword is related to the topics:

- **bayesian_calibration**
6.2. METHOD

Specification

Alias: none
Argument(s): INTEGER
Default: 1

Description
In DREAM, there is an extra adaptation during burn-in, in which DREAM estimates a distribution of crossover probabilities that favors large jumps over smaller ones in each of the chains. Normalization is required to ensure that all of the input dimensions contribute equally. In this process, a discrete number of candidate points for each crossover value is generated. This parameter is num_cr.

crossover_chain_pairs
- Keywords Area
- method
- bayes_calibration
- dream
- crossover_chain_pairs
  Number of chains used in crossover.

Topics
This keyword is related to the topics:
- bayesian_calibration

Specification

Alias: none
Argument(s): INTEGER
Default: 3

Description
The number of chains randomly selected to be used in the crossover each time a crossover occurs.

gr_threshold
- Keywords Area
- method
- bayes_calibration
- dream
- gr_threshold
  Convergence tolerance for the Gelman-Rubin statistic
Topics
This keyword is related to the topics:

- bayesian_calibration

Specification
Alias: none
Argument(s): REAL
Default: 1.2

Description
The gr_threshold is the convergence tolerance for the Gelman-Rubin statistic which will govern the convergence of the multiple chain process.

jump_step

- Keywords Area
- method
- bayes_calibration
- dream
- jump_step

Number of generations a long jump step is taken

Topics
This keyword is related to the topics:

- bayesian_calibration

Specification
Alias: none
Argument(s): INTEGER
Default: 5

Description
The integer jump_step forces a long jump every jump_step generations in DREAM.
emulator
- Keywords Area
- method
- bayes_calibration
- dream
- emulator

Use an emulator or surrogate model to evaluate the likelihood function

**Specification**

**Alias:** none  
**Argument(s):** none  

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required(Choose One)</td>
<td>emulator type (Group 1)</td>
<td>gaussian_process</td>
<td>Gaussian Process surrogate model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pce</td>
<td>Polynomial Chaos Expansion surrogate model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sc</td>
<td>Stochastic Collocation polynomial surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
</tbody>
</table>

**Description**

This keyword describes the type of emulator used when calculating the likelihood function for the Bayesian calibration. The emulator can be a Gaussian process, polynomial chaos expansion, or stochastic collocation.

**gaussian_process**
- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process

Gaussian Process surrogate model
### Specification

**Alias:** kriging  
**Argument(s):** none  
**Default:** Surfpack Gaussian process

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> (Choose One)</td>
<td></td>
<td>surfpack</td>
<td>Use the Surfpack version of Gaussian Process surrogates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dakota</td>
<td>Select the built in Gaussian Process surrogate</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td>emulator_samples</td>
<td></td>
<td>Number of data points used to train the surrogate model or emulator</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td>posterior_adaptive</td>
<td></td>
<td>Adapt the emulator model to achieve greater accuracy in regions of high posterior probability</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td>import_build_points_file</td>
<td></td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>

### Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the `surfpack` keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. For now, both versions are supported but the `dakota` version is deprecated and intended to be removed in a future release.

**surfpack**
- Keywords Area  
- method  
- bayes_calibration  
- dream  
- emulator  
- gaussian_process  
- surfpack

Use the Surfpack version of Gaussian Process surrogates
6.2. METHOD

Specification

Alias: none
Argument(s): none

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See optimization_method.

   The total number of evaluations of the likelihood function can be controlled using the max_trials keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See trend.

3. Correlation Lengths:

   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.

4. Ill-conditioning

   One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

   Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

   The surfpack model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

   The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

   See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also
significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

**dakota**

- **Keywords Area**
- **method**
- **bayes_calibration**
- **dream**
- **emulator**
- **gaussian_process**
- **dakota**

Select the built in Gaussian Process surrogate

**Specification**

*Alias*: none

*Argument(s):* none

**Description**

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the **gaussian_process** dakota model can be found in[58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a “nugget,” but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a **point_selection** option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.
6.2. METHOD

This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

emulator_samples

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- emulator_samples

Number of data points used to train the surrogate model or emulator

Specification
Alias: none
Argument(s): INTEGER

Description
This keyword refers to the number of build points or training points used to construct a Gaussian process emulator. If the user specifies a number of emulator_samples that is less than the minimum number of points required to build the GP surrogate, Dakota will augment the samples to obtain the minimum required.

posterior_adaptive

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- posterior_adaptive

Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

Specification
Alias: none
Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```
bayes_calibration queso
  samples = 2000  seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack emulator_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives
```

`import_build_points_file`

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- import_build_points_file

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom.annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>freeform</th>
<th>Selects freeform file format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>active_only</td>
</tr>
<tr>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

### Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

#### Default Behavior

By default, methods do not import points from a file.

#### Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

#### Examples

```plaintext
method polynomial_chaos
  expansion_order = 4
  import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- import_build_points_file
- annotated

Selects annotated tabular file format
```
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom.annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface   x1       x2     obj_fn nln_ineq_con_1 nln_ineq_con_2
1   NO_ID       0.9      1.1   0.0002    0.26     0.76
2   NO_ID  0.90009   1.1  0.0001996404857  0.2601620081  0.759955
3   NO_ID  0.89991   1.1  0.0002003604863  0.2598380081  0.760045
...```
6.2. METHOD

custom_annotated

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- import_build_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>header</td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td>eval_id</td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td>interface_id</td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```environment
  tabular_data
  tabular_data_file = 'dakota_summary.dat'
  custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

header

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated

eval_id

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no eval_id column

Description

See description of parent custom_annotated

interface_id

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
• `import_build_points_file`
• `custom_annotated`
• `interface_id`

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no interface_id column

**Description**

See description of parent `custom_annotated`

**freeform**

• `Keywords Area`
• `method`
• `bayes_calibration`
• `dream`
• `emulator`
• `gaussian_process`
• `import_build_points_file`
• `freeform`

Selects freeform file format

**Topics**

This keyword is related to the topics:

• `fileFormats`

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated format
6.2. **METHOD**

**Description**

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9   1.1    0.0002  0.26   0.76
0.90009 1.1  0.0001996404857  0.2601620081  0.759955
0.89991 1.1  0.0002003604863  0.2598380081  0.760045
...
```

**active_only**

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- gaussian_process
- import_build_points_file
- active_only

Import only active variables from tabular data file
Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword active_only indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

pce

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- pce

Polynomial Chaos Expansion surrogate model

Specification

Alias: none

Argument(s): none
6.2. METHOD

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong> (<em>Choose One</em>)</td>
<td><strong>Group 1</strong></td>
<td><strong>sparse_grid_level</strong></td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>expansion_order</strong></td>
<td>The (initial) order of a polynomial expansion</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>collocation_points</strong></td>
<td>Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.</td>
</tr>
</tbody>
</table>

**Description**

Selects a polynomial chaos expansion (PCE) surrogate model to use in the Bayesian likelihood calculations. When using PCE as a surrogate within the Bayesian framework, the PCE coefficients can be computed either from integration using a sparse grid or from regression using a random/unstructured data set.

**See Also**

These keywords may also be of interest:

- `polynomial_chaos`
- `sparse_grid_level`
- `Keywords Area`
- `method`
- `bayes_calibration`
- `dream`
- `emulator`
- `pce`
- `sparse_grid_level`

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation.
**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Description**

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non-nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product expansion rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.

- **expansion_order**
  - **Keywords Area**
  - method
  - bayes_calibration
  - dream
  - emulator
  - pce
  - expansion_order

  The (initial) order of a polynomial expansion

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST
Description

When the expansion order for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with expansion_samples). In this case, the expansion order $p$ cannot be inferred from the numerical integration specification and it is necessary to provide an expansion_order to specify $p$ for a total-order expansion.

Linear regression (specified with either collocation_points or collocation_ratio). A total-order expansion is used and must be specified using expansion_order as described in the previous option. To avoid requiring the user to calculate $N$ from $n$ and $p$, the collocation_ratio allows for specification of a constant factor applied to $N$ (e.g., collocation_ratio = 2. produces samples = $2N$). In addition, the default linear relationship with $N$ can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes $cN^o$ where $c$ is the collocation_ratio and $o$ is the ratio_order. The use_derivatives flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $cN^o$ + 1). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the tensor_grid keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If collocation_points or collocation_ratio is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved.
using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the noise_tolerance, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders (1,...,expansion_order) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

**collocation_ratio**

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- pce
- expansion_order
- collocation_ratio

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

**Specification**

**Alias:** none  
**Argument(s):** REAL

**Description**

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p, the collocation_ratio allows for specification of a constant factor applied to N (e.g., collocation_ratio = 2. produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes $cN^o$ where $c$ is the collocation_ratio and $o$ is the ratio_order. The use_derivatives flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).
6.2. METHOD

posterior_adaptive
• Keywords Area
• method
• bayes_calibration
• dream
• emulator
• pce
• expansion_order
• posterior_adaptive

Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

Specification

Alias: none
Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

bayes_calibration queso
  samples = 2000 seed = 348
delayed_rejection
  emulator
gaussian_process surfpack emulator_samples = 30
  posterior_adaptive max_iterations = 10
  proposal_covariance derivatives

import_build_points_file
• Keywords Area
• method
• bayes_calibration
• dream
• emulator
• pce
• expansion_order
• import_build_points_file

File containing points you wish to use to build a surrogate
**Specification**

**Alias:** import_points_file  
**Argument(s):** STRING  
**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional(Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**Examples**

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
  Keywords Area
  method
```
6.2. METHOD

- bayes_calibration
- dream
- emulator
- pce
- expansion_order
- import_build_points_file
- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
  Argument(s): none
  Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

- `custom_annotated`
  - Keywords Area
  - method
  - bayes_calibration
  - dream
  - emulator
  - pce
  - expansion_order
  - import_build_points_file
  - `custom_annotated`

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- `file_formats`

Specification

Alias: none

Argument(s): none

Default: annotated format
6.2. METHOD

### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

#### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

#### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

#### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id  x1  x2  obj_fn  nln_ineq_con_1  nln_ineq_con_2
1  0.9  1.1  0.0002  0.26  0.76
2  0.90009  1.1000000000 0.0001996404857  0.2601620081  0.759955
3  0.89991  1.1000000000 0.0002003604863  0.2598380081  0.760045
...```
header

• Keywords Area
• method
• bayes_calibration
• dream
• emulator
• pce
• expansion_order
• import_build_points_file
• custom_annotated
• header

Enable header row in custom-annotated tabular file

Specification

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated

eval_id

• Keywords Area
• method
• bayes_calibration
• dream
• emulator
• pce
• expansion_order
• import_build_points_file
• custom_annotated
• eval_id

Enable evaluation ID column in custom-annotated tabular file
6.2. METHOD

Specification

Alias: none
  Argument(s): none
  Default: no eval_id column

Description

See description of parent custom_annotated

interface_id
  • Keywords Area
  • method
  • bayes_calibration
  • dream
  • emulator
  • pce
  • expansion_order
  • import_build_points_file
  • custom_annotated
  • interface_id

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none
  Argument(s): none
  Default: no interface_id column

Description

See description of parent custom_annotated

freeform
  • Keywords Area
  • method
  • bayes_calibration
  • dream
  • emulator
- pce
- expansion_order
- import_build_points_file
- freeform

Selects freeform file format

Topics
This keyword is related to the topics:
- file formats

Specification
Alias: none
Argument(s): none
Default: annotated format

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```dakota
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
```

...
6.2. METHOD

active_only

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- pce
- expansion_order
- import_build_points_file
- active_only

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

collocation_points

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- pce
- collocation_points

Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

<table>
<thead>
<tr>
<th>Argument(s): INTEGERLIST</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>posterior_adaptive</td>
<td>Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
</tbody>
</table>

Description

Specify the number of collocation points used to estimate PCE coefficients using regression or orthogonal-least-interpolation.

posterior_adaptive
  - Keywords Area
  - method
  - bayes_calibration
  - dream
  - emulator
  - pce
  - collocation_points
  - posterior_adaptive

Adapt the emulator model to achieve greater accuracy in regions of high posterior probability.

Specification

Alias: none

<table>
<thead>
<tr>
<th>Argument(s): none</th>
</tr>
</thead>
</table>

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.
6.2. METHOD

Examples

```plaintext
bayes_calibration queso
  samples = 2000 seed = 348
delayed_rejection
emulator
gaussian_process surfpac  emulator_samples = 30
posterior_adaptive max_iterations = 10
proposal_covariance derivatives
```

import_build_points_file

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- pce
- collocation_points

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom_annotated
- freeform

**Examples**

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated
- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- pce
- collocation_points
- import_build_points_file
- annotated

Selects annotated tabular file format
```

**Topics**

This keyword is related to the topics:

- file_formats
6.2. **METHOD**

**Specification**

*Alias:* none  
*Argument(s):* none  
*Default:* annotated format

**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The *annotated* keyword can be used to explicitly specify this.

**Usage Tips**

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify `custom_annotated`.
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to *annotated* format, though *freeform* remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the *freeform* option.

**Examples**

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
$eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

`custom_annotated`

- **Keywords Area**
- **method**
- **bayes_calibration**
- **dream**
- **emulator**
• pce
• collocation_points
• import_build_points_file
• custom_annotated

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:
• file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description
A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior
The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips
• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id x1   x2   obj_fn  nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002  0.26  0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...%
```

header

• Keywords Area
• method
• bayes_calibration
• dream
• emulator
• pce
• collocation_points
• import_build_points_file
• custom_annotated
• header

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent custom_annotated
**eval_id**
- **Keywords Area**
- method
- bayes_calibration
- dream
- emulator
- pce
- collocation_points
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent custom_annotated

**interface_id**
- **Keywords Area**
- method
- bayes_calibration
- dream
- emulator
- pce
- collocation_points
- import_build_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file
6.2. METHOD

### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** no interface_id column

### Description

See description of parent `custom.annotated`

- `freeform`
  - **Keywords Area**
  - method
  - bayes.calibration
  - dream
  - emulator
  - pce
  - collocation.points
  - import.build.points_file
  - `freeform`

Selects freeform file format

### Topics

This keyword is related to the topics:

- `file_formats`

### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** annotated format

### Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The annotated format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

**Usage Tips**
Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.

When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9 1.1 0.0002 0.26 0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

active_only

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- pce
- collocation_points
- import_build_points_file
- active_only

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- file_formats
6.2. **METHOD**

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

```plaintext
sc
```

- **Keywords Area**
- **method**
- **bayes_calibration**
- **dream**
- **emulator**
- **sc**

Stochastic Collocation polynomial surrogate model

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td><code>sparse_grid_level</code></td>
<td>Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation</td>
</tr>
</tbody>
</table>

**Description**

Selects stochastic collocation (SC) model to use in the Bayesian likelihood calculations. When using SC as a surrogate within the Bayesian framework, the build points (training points) for the stochastic collocation are constructed from a sparse grid.
See Also

These keywords may also be of interest:

- stoch_collocation

sparse_grid_level

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- sc
- sparse_grid_level

Set the sparse grid level to be used when performing sparse grid integration or sparse grid interpolation.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with sparse_grid_level and, optionally, dimension_preference). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit non_nested specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the dimension_preference specification enables the use of anisotropic sparse grids (refer to the PCE description in the User’s Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full sparse_grid_level and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User’s Manual for additional details). As for quadrature_order, the sparse_grid_level specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

This keyword can be used when using sparse grid integration to calculate PCE coefficients or when generating samples for sparse grid collocation.
6.2. METHOD

use_derivatives

- Keywords Area
- method
- bayes_calibration
- dream
- emulator
- use_derivatives

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The use_derivatives flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it’s use with Surfpack Gaussian process is not recommended.

standardized_space

- Keywords Area
- method
- bayes_calibration
- standardized_space

Perform Bayesian inference in standardized probability space

Specification

Alias: none

Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

This option transforms the inference process (MCMC sampling and any emulator model management) into a standardized probability space.

The variable transformations performed are as described in askey.

Default Behavior

The default for the Gaussian process and no emulator options is to perform inference in the original probability space (no transformation). Polynomial chaos and stochastic collocation emulators, on the other hand, are always formed in standardized probability space, such that the inference process is also performed in this standardized space.

Expected Output

The user will see the truth model evaluations performed in the original space, whereas any method diagnostics relating to the MCMC samples (e.g., QUESO data in the outputData directory) will report points and response data (response gradients and Hessians, if present, will differ but response values will not) that correspond to the transformed space.

Usage Tips

Selecting standardized_space generally has the effect of scaling the random variables to be of comparable magnitude, which can improve the efficiency of the Bayesian inference process.

Examples

```plaintext
method,
    bayes_calibration queso
    samples = 2000 seed = 348
dram
    standardized_space
```

calibrate_error_multipliers

- Keywords Area
- method
- bayes_calibration
- calibrate_error_multipliers

Calibrate hyper-parameter multipliers on the observation error covariance

Specification

Alias: none

Argument(s): none

Default: none
### 6.2. METHOD

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>one</td>
<td>Calibrate one hyper-parameter multiplier across all responses/experiments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>per_experiment</td>
<td>Calibrate one hyper-parameter multiplier per experiment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>per_response</td>
<td>Calibrate one hyper-parameter multiplier per response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>both</td>
<td>Calibrate one hyper-parameter multiplier for each response/experiment pair</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>hyperprior_alphas</td>
<td>Shape (alpha) of the inverse gamma hyper-parameter prior</td>
</tr>
</tbody>
</table>

**Description**

Calibrate one or more multipliers on the user-provided observation error covariance (\texttt{variance_type}). Options include one multiplier on the whole block-diagonal covariance structure, one multiplier per experiment covariance block, one multiplier per response covariance block, or separate multipliers for each response/experiment pair (for a total of number experiments $\times$ number response groups).

**Default Behavior:** No hyper-parameter calibration. When hyper-parameter calibration is enabled, the default prior on the multiplier is a diffuse inverse gamma, with mean and mode approximately 1.0.

**Expected Output:** Final calibration results will include both inference parameters and one or more calibrated hyper-parameters.

**Usage Tips:** The per_response option can be useful when each response has its own measurement error process, but all experiments were gathered with the same equipment and conditions. The per_experiment option might be used when working with data from multiple independent laboratories.

**Examples**

Perform Bayesian calibration with 2 calibration variables and two hyper-parameter multipliers, one per each of two responses. The multipliers are assumed the same across the 10 experiments. The priors on the multipliers are specified using the \texttt{hyperprior_alphas} and \texttt{hyperprior_betas} keywords.
bayes_calibration queso
samples = 1000 seed = 348
dram
calibrate_error_multipliers per_response
hyperprior_alphas = 27.0
hyperprior_betas = 26.0

variables
uniform_uncertain 2
upper_bounds 1.e8 10.0
lower_bounds 1.e6 0.1
initial_point 2.85e7 2.5
descriptors 'E' 'w'

responses
 calibration_terms = 2
calibration_data_file = 'expdata.withsigma.dat'
 freeform
  num_experiments = 10
  variance_type = 'scalar'

one

  • Keywords Area
  • method
  • bayes_calibration
  • calibrate_error_multipliers
  • one

  Calibrate one hyper-parameter multiplier across all responses/experiments

Specification

Alias: none
Argument(s): none

Description

A single hyper-parameter multiplying all response/experiment covariance blocks will be estimated.

per_experiment

  • Keywords Area
  • method
  • bayes_calibration
  • calibrate_error_multipliers
  • per_experiment

  Calibrate one hyper-parameter multiplier per experiment
6.2. METHOD

**Specification**

Alias: none

Argument(s): none

**Description**

One hyper-parameter multiplying each experiment covariance block will be estimated.

**per_response**

- Keywords Area
- method
- bayes_calibration
- calibrate_error_multipliers
- per_response
  
  Calibrate one hyper-parameter multiplier per response

**Specification**

Alias: none

Argument(s): none

**Description**

One hyper-parameter multiplying each response covariance block will be estimated.

**both**

- Keywords Area
- method
- bayes_calibration
- calibrate_error_multipliers
- both
  
  Calibrate one hyper-parameter multiplier for each response/experiment pair

**Specification**

Alias: none

Argument(s): none

**Description**

One hyper-parameter multiplying each experiment/response covariance block will be estimated.
**hyperprior_alphas**

- Keywords Area
- method
- bayes_calibration
- calibrate_error_multipliers
- hyperprior_alphas

Shape (alpha) of the inverse gamma hyper-parameter prior

**Specification**

**Alias:** none

**Argument(s):** REALLIST

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>hyperprior_betas</td>
<td>Scale (beta) of the inverse gamma hyper-parameter prior</td>
</tr>
</tbody>
</table>

**Description**

Shape of the prior distribution for calibrated error multipliers. Either a single value or number of hyper-parameters values may be specified.

**Default:** 102.0 (with beta = 103.0) so mean and mode are approximately 1.0 and standard deviation is about 0.1.

**hyperprior_betas**

- Keywords Area
- method
- bayes_calibration
- calibrate_error_multipliers
- hyperprior_alphas
- hyperprior_betas

Scale (beta) of the inverse gamma hyper-parameter prior

**Specification**

**Alias:** none

**Argument(s):** REALLIST
6.2. METHOD

Description

Scale of the prior distribution for calibrated error multipliers. Either a single value or number of hyper-parameters values may be specified.

Defaults to 103.0 (with alpha = 102.0) so mean and mode are approximately 1.0 and standard deviation is about 0.1.

convergence_tolerance

- **Keywords Area**
- method
- bayes_calibration
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

**Alias:** none

**Argument(s):** REAL

**Default:** 1.e-4

Description

The *convergence_tolerance* specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
• NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

• NL2SOL: See nl2sol

max_iterations

• Keywords Area
• method
• bayes_calibration
• max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

• method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence}$/ efficient_global: 25∗n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

samples

• Keywords Area
• method
• bayes_calibration
• samples

Number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: method-dependent
6.2. METHOD

Description
The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior
By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips
To obtain linear sensitivities or to construct a linear response surface, at least \( \text{dim} + 1 \) samples should be used, where \( \text{dim} \) is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least \( \frac{(\text{dim}+1)(\text{dim}+2)}{2} \) samples are needed. For uncertainty quantification, we recommend at least \( 10 \times \text{dim} \) samples. For variance-based decomps, we recommend hundreds to thousands of samples. Note that for variance-based decomps, the number of simulations performed will be \( N \times (\text{dim}+2) \).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

```
seed
```

- Keywords Area
- method
- bayes_calibration
- seed

Seed of the random number generator

Specification

Alias: none
Argument(s): INTEGER
Default: system-generated (non-repeatable)

Description
The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.
Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

model_pointer

- Keywords Area
- method
- bayes_calibration
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none
  Argument(s): STRING
  Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.
  See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = ‘UQ’

method
  id_method = ‘UQ’
  model_pointer = ‘SURR’
  sampling,
    samples = 10
6.2. METHOD

seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.58 dace

- Keywords Area
- method
- dace

Design and Analysis of Computer Experiments

Topics

This keyword is related to the topics:

- package_ddace
- design_and_analysis_of_computer_experiments
### Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>DACE type (Group 1)</td>
<td>random</td>
<td>Uses purely random Monte Carlo sampling to sample variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>oas</td>
<td>Orthogonal Array Sampling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lhs</td>
<td>Uses Latin Hypercube Sampling (LHS) to sample variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>oa_lhs</td>
<td>Orthogonal Array Latin Hypercube Sampling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>box_behnken</td>
<td>Box-Behnken Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>central_composite</td>
<td>Central Composite Design</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>main_effects</td>
<td>ANOVA</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>quality_metrics</td>
<td>Calculate metrics to assess the quality of quasi-Monte Carlo samples</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>variance_based_decomp</td>
<td>Activates global sensitivity analysis based on decomposition of response variance into contributions from variables</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>fixed_seed</th>
<th>Reuses the same seed value for multiple random sampling sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>symbols</td>
<td>Number of replications in the sample set</td>
</tr>
<tr>
<td>Optional</td>
<td>samples</td>
<td>Number of samples for sampling-based methods</td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

The Distributed Design and Analysis of Computer Experiments (DDACE) library provides the following DACE techniques:

1. grid sampling (grid)
2. pure random sampling (random)
3. orthogonal array sampling (oas)
4. latin hypercube sampling (lhs)
5. orthogonal array latin hypercube sampling (oa_lhs)
6. Box-Behnken (box_benhken)
7. central composite design (central_composite)

These methods all generate point sets that may be used to drive a set of computer experiments. Note that all of the DACE methods generated randomized designs, except for Box-Behnken and Central composite which are classical designs. That is, the grid sampling will generate a randomized grid, not what one typically thinks of as a grid of uniformly spaced points over a rectangular grid. Similar, the orthogonal array is a randomized version of an orthogonal array: it does not generate discrete, fixed levels.

In addition to the selection of the method, there are keywords that affect the method outputs:

1. main_effects
2. quality_metrics
3. variance_based_decomp

And keywords that affect the sampling:
1. fixed_seed
2. symbols
3. samples
4. seed

See Also

These keywords may also be of interest:

- fsu_cvt
- fsu_quasi_mc
- psuade_moat

grid

- Keywords Area
- method
- dace
- grid

Grid Sampling

Specification

Alias: none
Argument(s): none

Description

The grid option in DACE will produce a randomized grid of points. If you are interested in a regular grid of points, use the multidimensional parameter study (under Parameter Studies) instead. Grid Sampling

random

- Keywords Area
- method
- dace
- random

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none
Argument(s): none
6.2. **METHOD**

**Description**

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

**Default Behavior**

Monte Carlo sampling is not used by default. To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

**Usage Tips**

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

**Examples**

```
method sampling
    sample_type random
    samples = 200
```

**oas**

- **Keywords Area**

- **method**

- **dace**

- **oas**

Orthogonal Array Sampling

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Orthogonal array sampling (OAS) is a widely used technique for running experiments and systematically testing factor effects. An orthogonal array sample can be described as a 4-tuple \((m; n; s; r)\), where \(m\) is the number of sample points, \(n\) is the number of input variables, \(s\) is the number of symbols, and \(r\) is the strength of the orthogonal array. The number of sample points, \(m\), must be a multiple of the number of symbols, \(s\). The number of symbols refers to the number of levels per input variable. The strength refers to the number of columns where we are guaranteed to see all the possibilities an equal number of times. Note that the DACE OAS capability produces a randomized orthogonal array: the samples for a particular level are randomized within that level.

If one examines the sample sets in an orthogonal array by looking at the rows as individual samples and columns as the variables sampled, one sees that the columns are orthogonal to each other in an orthogonal array. This feature is important in main effects analysis, which is a sensitivity analysis technique that identifies which variables have the most influence on the output.
Uses Latin Hypercube Sampling (LHS) to sample variables

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

**Default Behavior**

By default, Latin Hypercube Sampling is used. To explicitly specify this in the Dakota input file, however, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

**Usage Tips**

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

**Examples**

```
method
  sampling
    sample_type lhs
    samples = 20
```

**oa_lhs**

**Specification**

**Alias:** none

**Argument(s):** none
6.2. METHOD

Description

The Orthogonal Array Latin Hypercube Sampling option in DACE produces a “latinized” version of an orthogonal array. That is, after the orthogonal array is generated, the samples go through a stratification process to produce samples that have been both orthogonalized and stratified.

See Also

These keywords may also be of interest:

- oas

box_behnken

Keywords Area

- method
- dace
- box_behnken

Box-Behnken Design

Specification

Alias: none

Argument(s): none

Description

The Box-Behnken design is similar to a Central Composite design, with some differences. The Box-Behnken design is a quadratic design in that it does not contain an embedded factorial or fractional factorial design. In this design the treatment combinations are at the midpoints of edges of the process space and at the center, as compared with CCD designs where the extra points are placed at star points on a circle outside of the process space. Box-Behnken designs are rotatable (or near rotatable) and require 3 levels of each factor.

central_composite

Keywords Area

- method
- dace
- central_composite

Central Composite Design

Specification

Alias: none

Argument(s): none
Description

A central composite design (CCD), contains an embedded factorial or fractional factorial design with a center points that is augmented with a group of “star points” that allow estimation of curvature.

Examples

See the User’s Manual for an example.

main_effects

- Keywords Area
- method
- dace
- main_effects

ANOVA

Specification

Alias: none
Argument(s): none
Default: No main_effects

Description

The main_effects control prints Analysis-of-Variance main effects results (e.g. ANOVA tables with p-values per variable). The main_effects control is only operational with the orthogonal arrays or Latin Hypercube designs, not for Box Behnken or Central Composite designs.

Main effects is a sensitivity analysis method which identifies the input variables that have the most influence on the output. In main effects, the idea is to look at the mean of the response function when variable A (for example) is at level 1 vs. when variable A is at level 2 or level 3. If these mean responses of the output are statistically significantly different at different levels of variable A, this is an indication that variable A has a significant effect on the response. The orthogonality of the columns is critical in performing main effects analysis, since the column orthogonality means that the effects of the other variables “cancel out” when looking at the overall effect from one variable at its different levels.

quality_metrics

- Keywords Area
- method
- dace
- quality_metrics

Calculate metrics to assess the quality of quasi-Monte Carlo samples
6.2. **METHOD**

**Topics**

This keyword is related to the topics:

- package_fsudace

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** No quality_metrics

**Description**

`quality_metrics` calculates four quality metrics relating to the volumetric spacing of the samples. The four quality metrics measure different aspects relating to the uniformity of point samples in hypercubes. Desirable properties of such point samples are:

- are the points equally spaced
- do the points cover the region
- and are they isotropically distributed
- with no directional bias in the spacing

The four quality metrics we report are:

- h: the point distribution norm, which is a measure of uniformity of the point distribution
- chi: a regularity measure, and provides a measure of local uniformity of a set of points
- tau: the second moment trace measure
- d: the second moment determinant measure

All of these values are scaled so that smaller is better (the smaller the metric, the better the uniformity of the point distribution).

**Examples**

Complete explanation of these measures can be found in [38].

**variance_based_decomp**

- **Keywords Area**
- **method**
- **dace**
- **variance_based_decomp**

Activates global sensitivity analysis based on decomposition of response variance into contributions from variables.
CHAPTER 6. KEYWORDS AREA

Specification

**Alias:** none

**Argument(s):** none

**Default:** no variance-based decomposition

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><em>drop_tolerance</em></td>
<td>Suppresses output of sensitivity indices with values lower than this tolerance</td>
</tr>
</tbody>
</table>

Description

Dakota can calculate sensitivity indices through variance based decomposition using the keyword *variance-based_decomp*. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

**Default Behavior**

Because of the computational cost, *variance_based_decomp* is turned off as a default.

If the user specified a number of samples, N, and a number of nondeterministic variables, M, variance-based decomposition requires the evaluation of N*(M+2) samples. **Note that specifying this keyword will increase the number of function evaluations above the number requested with the samples keyword since replicated sets of sample values are evaluated.**

**Expected Outputs**

When *variance_based_decomp* is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response.

**Usage Tips**

To obtain sensitivity indices that are reasonably accurate, we recommend that N, the number of samples, be at least one hundred and preferably several hundred or thousands.

Examples

```python
method,
sampling
    sampling_type lhs
    samples = 100
    variance_based_decomp
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of[73]: “The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input.”

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, $S_i$, means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in[73] and[87].
6.2. METHOD

**drop_tolerance**

- Keywords Area
- method
- dace
- variance_based_decomp
- drop_tolerance

Suppresses output of sensitivity indices with values lower than this tolerance

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** All VBD indices displayed

**Description**

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

**Default Behavior**

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

**Usage Tips**

For polynomial chaos, which outputs main, interaction, and total effects by default, the univariate-effects may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

**Examples**

```plaintext
method,
sampling
    sample_type lhs
    samples = 100
    variance_based_decomp
drop_tolerance = 0.001
```

**fixed_seed**

- Keywords Area
- method
- dace
- fixed_seed

Reuses the same seed value for multiple random sampling sets
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none
Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method sampling
   sample_type lhs
   samples = 10
   fixed_seed
```

symbols

- Keywords Area
- method
- dace
- symbols

Number of replications in the sample set

Specification

Alias: none
Argument(s): INTEGER
Default: default for sampling algorithm

Description

`symbols` is related to the number of levels per variable in the sample set (a larger number of symbols equates to more stratification and fewer replications). For example, if `symbols = 7`, each variable would be divided into seven levels.
6.2. METHOD

samples

- Keywords Area
- method
- dace
- samples

Number of samples for sampling-based methods

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** method-dependent

**Description**

The *samples* keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance-based decomp, we recommend hundreds to thousands of samples. Note that for variance-based decomp, the number of simulations performed will be N*(dim+2).

**Examples**

```
method sampling
   sample_type lhs
   samples = 20
```

seed

- Keywords Area
- method
- dace
- seed

Seed of the random number generator

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** system-generated (non-repeatable)
Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

model_pointer
```

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.
6.2. METHOD

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.
See `block_pointer` for details about pointers.

Examples

environment
   tabular_graphics_data
   method_pointer = 'UQ'

method
   id_method = 'UQ'
   model_pointer = 'SURR'
   sampling,
   samples = 10
   seed = 98765 rng rnum2
   response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
   sample_type lhs
   distribution cumulative

model
   id_model = 'SURR'
   surrogate global,
   dace_method_pointer = 'DACE'
   polynomial quadratic

method
   id_method = 'DACE'
   model_pointer = 'DACE_M'
   sampling sample_type lhs
   samples = 121 seed = 5034 rng rnum2

model
   id_model = 'DACE_M'
   single
   interface_pointer = 'I1'

variables
   uniform_uncertain = 2
   lower_bounds = 0. 0.
   upper_bounds = 1. 1.
   descriptors = 'x1' 'x2'

interface
   id_interface = 'I1'
   system asynch evaluation_concurrency = 5
   analysis_driver = 'text_book'

responses
   response_functions = 3
   no_gradients
   no_hessians

6.2.59 fsu_cvt

- Keywords Area
Design of Computer Experiments - Centroidal Voronoi Tessellation

**Topics**
This keyword is related to the topics:

- package_fsudace
- design_and_analysis_of_computer_experiments

### Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>latinize</td>
<td>Adjust samples to improve the discrepancy of the marginal distributions</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>quality_metrics</td>
<td>Calculate metrics to assess the quality of quasi-Monte Carlo samples</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>variance_based_decomp</td>
<td>Activates global sensitivity analysis based on decomposition of response variance into contributions from variables</td>
</tr>
</tbody>
</table>
6.2. METHOD

| Optional       | fixed_seed  | Reuses the same seed value for multiple random sampling sets |
|                |            |                                                             |
| Optional       | trial_type | Specify how the trial samples are generated                 |
| Optional       | num_trials | The number of secondary sample points generated to adjust the location of the primary sample points |
| Optional       | max_iterations | Stopping criterion based on number of iterations |
| Optional       | samples    | Number of samples for sampling-based methods |
| Optional       | seed       | Seed of the random number generator |
| Optional       | model_pointer | Identifier for model block to be used by a method |

**Description**

The FSU Centroidal Voronoi Tessellation method (fsu_cvt) produces a set of sample points that are (approximately) a Centroidal Voronoi Tessellation. The primary feature of such a set of points is that they have good volumetric spacing; the points tend to arrange themselves in a pattern of cells that are roughly the same shape.

To produce this set of points, an almost arbitrary set of initial points is chosen, and then an internal set of iterations is carried out. These iterations repeatedly replace the current set of sample points by an estimate of the centroids of the corresponding Voronoi subregions. [18].

The user may generally ignore the details of this internal iteration. If control is desired, however, there are a few variables with which the user can influence the iteration. The user may specify:

- **max_iterations**, the number of iterations carried out
- **num_trials**, the number of secondary sample points generated to adjust the location of the primary sample points
- **trial_type**, which controls how these secondary sample points are generated

This method generates sets of uniform random variables on the interval [0,1]. If the user specifies lower and upper bounds for a variable, the [0,1] samples are mapped to the [lower, upper] interval.
Theory

This method is designed to generate samples with the goal of low discrepancy. Discrepancy refers to the nonuniformity of the sample points within the hypercube.

Discrepancy is defined as the difference between the actual number and the expected number of points one would expect in a particular set $B$ (such as a hyper-rectangle within the unit hypercube), maximized over all such sets. Low discrepancy sequences tend to cover the unit hypercube reasonably uniformly.

Centroidal Voronoi Tessellation does very well volumetrically: it spaces the points fairly equally throughout the space, so that the points cover the region and are isotropically distributed with no directional bias in the point placement. There are various measures of volumetric uniformity which take into account the distances between pairs of points, regularity measures, etc. Note that Centroidal Voronoi Tessellation does not produce low-discrepancy sequences in lower dimensions. The lower-dimension (such as 1-D) projections of Centroidal Voronoi Tessellation can have high discrepancy.

See Also

These keywords may also be of interest:

- dace
- fsu_quasi_mc
- psuade_moat

latinize

- Keywords Area
- method
- fsu_cvt
- latinize

Adjust samples to improve the discrepancy of the marginal distributions

Specification

Alias: none

Argument(s): none

Default: No latinization

Description

The latinize control takes the samples and "latinizes" them, meaning that each original sample is moved so that it falls into one strata or bin in each dimension as in Latin Hypercube sampling. The default setting is NOT to latinize. However, one may be interested in doing this in situations where one wants better discrepancy of the 1-dimensional projections (the marginal distributions).
6.2. METHOD

quality_metrics

- Keywords Area
- method
- fsu_cvt
- quality_metrics

Calculate metrics to assess the quality of quasi-Monte Carlo samples

Topics

This keyword is related to the topics:

- package_fsudace

Specification

Alias: none

Argument(s): none

Default: No quality_metrics

Description

quality_metrics calculates four quality metrics relating to the volumetric spacing of the samples. The four quality metrics measure different aspects relating to the uniformity of point samples in hypercubes. Desirable properties of such point samples are:

- are the points equally spaced
- do the points cover the region
- and are they isotropically distributed
- with no directional bias in the spacing

The four quality metrics we report are:

- h: the point distribution norm, which is a measure of uniformity of the point distribution
- chi: a regularity measure, and provides a measure of local uniformity of a set of points
- tau: the second moment trace measure
- d: the second moment determinant measure

All of these values are scaled so that smaller is better (the smaller the metric, the better the uniformity of the point distribution).

Examples

Complete explanation of these measures can be found in [38].
 Activates global sensitivity analysis based on decomposition of response variance into contributions from variables

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no variance-based decomposition

| Required/- | Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional | | drop_tolerance | Suppresses output of sensitivity indices with values lower than this tolerance |

| Required/- | Description of Group | Dakota Keyword | Dakota Keyword Description |
| Optional | | | |

**Description**

Dakota can calculate sensitivity indices through variance based decomposition using the keyword `variance_based_decomp`. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

**Default Behavior**

Because of the computational cost, `variance_based_decomp` is turned off as a default.

If the user specified a number of samples, \( N \), and a number of nondeterministic variables, \( M \), variance-based decomposition requires the evaluation of \( N(M + 2) \) samples. Note that specifying this keyword will increase the number of function evaluations above the number requested with the `samples` keyword since replicated sets of sample values are evaluated.

**Expected Outputs**

When `variance_based_decomp` is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response.

**Usage Tips**

To obtain sensitivity indices that are reasonably accurate, we recommend that \( N \), the number of samples, be at least one hundred and preferably several hundred or thousands.

**Examples**

```
method,
sampling
   sample_type lhs
```
6.2. METHOD

samples = 100
variance_based_decomp

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of[73]: “The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input.”

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, $S_i$, means that the uncertainty in the input variable $i$ has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in[73] and[87].

drop_tolerance

- Keywords Area
- method
- fsu_cvt
- variance_based_decomp
- drop_tolerance

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The drop_tolerance keyword allows the user to specify a value below which sensitivity indices generated by variance_based_decomp are not displayed.

Default Behavior

By default, all sensitivity indices generated by variance_based_decomp are displayed.

Usage Tips

For polynomial_chaos, which outputs main, interaction, and total effects by default, the univariate_effects may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).
Examples

```plaintext
method,
sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

**fixed_seed**

- **Keywords Area**
- **method**
- **fsu_cvt**
- **fixed_seed**

Reuses the same seed value for multiple random sampling sets

**Specification**

Alias: none

- **Argument(s):** none
- **Default:** not fixed; pattern varies run-to-run

**Description**

The *fixed_seed* flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

**Default Behavior**

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the *seed* must also be specified.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

**trial_type**

- **Keywords Area**
- **method**
- **fsu_cvt**
- **trial_type**

Specify how the trial samples are generated
6.2. METHOD

Specification

Alias: none
Argument(s): none
Default: random

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trial type (Group 1)</td>
<td>grid</td>
<td>halton</td>
<td>Generate samples from a Halton sequence</td>
</tr>
<tr>
<td>grid</td>
<td>random</td>
<td>random</td>
<td>Uses purely random Monte Carlo sampling to sample variables</td>
</tr>
</tbody>
</table>

Description

The user has the option to specify the method by which the trials are created to adjust the centroids. The `trial_type` can be one of three types:

- `random`, where points are generated randomly
- `halton`, where points are generated according to the Halton sequence
- `grid`, where points are placed on a regular grid over the hyperspace.

grid

- Keywords Area
- method
- fsu_cvt
- trial_type
- grid

Samples on a regular grid

Specification

Alias: none
Argument(s): none

Description

Points are placed on a regular grid over the hyperspace.
See Also
These keywords may also be of interest:

- trial_type

halton

- Keywords Area
- method
- fsu_cvt
- trial_type
- halton

Generate samples from a Halton sequence

Topics
This keyword is related to the topics:

- package_fsudace

Specification
Alias: none
Argument(s): none

Description
The quasi-Monte Carlo sequences of Halton are deterministic sequences determined by a set of prime bases. These sequences generate random numbers with the goal of filling a unit hypercube uniformly.

Generally, we recommend that the user leave the default setting for the bases, which are the lowest primes. Thus, if one wants to generate a sample set for 3 random variables, the default bases used are 2, 3, and 5 in the Halton sequence. To give an example of how these sequences look, the Halton sequence in base 2 starts with points 0.5, 0.25, 0.75, 0.125, 0.625, etc. The first few points in a Halton base 3 sequence are 0.33333, 0.66667, 0.11111, 0.44444, 0.77777, etc. Notice that the Halton sequence tends to alternate back and forth, generating a point closer to zero then a point closer to one. An individual sequence is based on a radix inverse function defined on a prime base. The prime base determines how quickly the [0,1] interval is filled in.

Theory
For more information about these sequences, see [43], [44], and [1].
6.2. METHOD

random

- **Keywords Area**
- **method**
- **fsu_cvt**
- **trial_type**
- **random**

Uses purely random Monte Carlo sampling to sample variables

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

**Default Behavior**

Monte Carlo sampling is not used by default. To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

**Usage Tips**

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

**Examples**

```
method
  sampling
    sample_type random
    samples = 200
```

**num_trials**

- **Keywords Area**
- **method**
- **fsu_cvt**
- **num_trials**

The number of secondary sample points generated to adjust the location of the primary sample points

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 10000
Description
In general, the variable with the most influence on the quality of the final sample set is num_trials, which determines how well the Voronoi subregions are sampled.

Generally, num_trials should be "large", certainly much bigger than the number of sample points being requested; a reasonable value might be 10,000, but values of 100,000 or 1 million are not unusual.

max_iterations

- Keywords Area
- method
- fsu_cvt
- max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:

- methodIndependentControls

Specification
Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global{reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

samples

- Keywords Area
- method
- fsu_cvt
- samples

Number of samples for sampling-based methods

Specification
Alias: none

Argument(s): INTEGER

Default: method-dependent
6.2. METHOD

Description

The samples keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior
By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips
To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance_based_decomp, we recommend hundreds to thousands of samples. Note that for variance_based_decomp, the number of simulations performed will be N*(dim+2).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20

seed
  
  • Keywords Area

  • method

  • fsu_cvt

  • seed

  Seed of the random number generator
```

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.
Examples

method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347

model_pointer

  • Keywords Area
  • method
  • fsu_cvt
  • model_pointer

  Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

  • block_pointer

Specification

Alias: none

  Argument(s): STRING
  Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

  Default Behavior
  If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

  Usage Tips
  When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.
  See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
6.2. METHOD

seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
                0.1 0.2 0.6
                0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
  id_model = 'SURRE'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.60 psuade_moat

  • Keywords Area
  • method
  • psuade_moat
  Morris One-at-a-Time

Topics
This keyword is related to the topics:

  • package_psuade
  • design_and_analysis_of_computer_experiments
**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>partitions</td>
<td>partitions</td>
<td>Number of partitions of each variable</td>
</tr>
<tr>
<td>Optional</td>
<td>samples</td>
<td>samples</td>
<td>Number of samples for sampling-based methods</td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>seed</td>
<td>Seed of the random number generator</td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

The Morris One-At-A-Time (MOAT) method, originally proposed by Morris [62], is a screening method, designed to explore a computational model to distinguish between input variables that have negligible, linear and additive, or nonlinear or interaction effects on the output. The computer experiments performed consist of individually randomized designs which vary one input factor at a time to create a sample of its elementary effects.

The number of samples (samples) must be a positive integer multiple of (number of continuous design variable + 1) and will be automatically adjusted if misspecified.

The number of partitions (partitions) applies to each variable being studied and must be odd (the number of MOAT levels per variable is partitions + 1). This will also be adjusted at runtime as necessary.

For information on practical use of this method, see [73].

**Theory**

With MOAT, each dimension of a $k$-dimensional input space is uniformly partitioned into $p$ levels, creating a grid of $p^k$ points $x \in \mathbb{R}^k$ at which evaluations of the model $g(x)$ might take place. An elementary effect corresponding to input $i$ is computed by a forward difference

$$d_i(x) = \frac{g(x + \Delta e_i) - g(x)}{\Delta},$$

where $e_i$ is the $i$th coordinate vector, and the step $\Delta$ is typically taken to be large (this is not intended to be a local derivative approximation). In the present implementation of MOAT, for an input variable scaled to $[0, 1]$, $\Delta = \frac{p}{2(p-1)}$, so the step used to find elementary effects is slightly larger than half the input range.

The distribution of elementary effects $d_i$ over the input space characterizes the effect of input $i$ on the output of interest. After generating $r$ samples from this distribution, their mean,

$$\mu_i = \frac{1}{r} \sum_{j=1}^{r} d_i^{(j)}$$
modified mean

\[
\mu_i^* = \frac{1}{r} \sum_{j=1}^{r} |d_{i}^{(j)}|,
\]

(using absolute value) and standard deviation

\[
\sigma_i = \sqrt{\frac{1}{r} \sum_{j=1}^{r} (d_{i}^{(j)} - \mu_i)^2}
\]

are computed for each input \(i\). The mean and modified mean give an indication of the overall effect of an input on the output. Standard deviation indicates nonlinear effects or interactions, since it is an indicator of elementary effects varying throughout the input space.

**partitions**

- **Keywords Area**
- **method**
- **psuade_moat**
- **partitions**

Number of partitions of each variable

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** 3

**Description**

Described on the parent page, **psuade_moat**

**samples**

- **Keywords Area**
- **method**
- **psuade_moat**
- **samples**

Number of samples for sampling-based methods

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** method-dependent
CHAPTER 6. KEYWORDS AREA

Description

The samples keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least dim+1 samples should be used, where “dim” is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least (dim+1)(dim+2)/2 samples are needed. For uncertainty quantification, we recommend at least 10*dim samples. For variance_based_decomp, we recommend hundreds to thousands of samples. Note that for variance_based_decomp, the number of simulations performed will be N*(dim+2).

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 20

seed

  • Keywords Area
  • method
  • psuade_moat
  • seed

Seed of the random number generator
```

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.
6.2. METHOD

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**model_pointer**

- **Keywords Area**
- **method**
- **psuade_moat**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The **model_pointer** is used to specify which **model** block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a **model** block in the Dakota input file that has a corresponding **id_model** with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a **model_pointer** for each method is imperative.

See **block_pointer** for details about pointers.

**Examples**

```plaintext
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
```
6.2.61  local_evidence

- Keywords Area
- method
- local_evidence

Evidence theory with evidence measures computed with local optimization methods

Topics
This keyword is related to the topics:

- epistemic_uncertainty_quantification_methods
- evidence_theory
### 6.2. METHOD

**Specification**

**Alias:** nond_local_evidence

**Argument(s):** none

| Required/- |
| Optional |
| Optional/Choose One | Description of Group | Dakota Keyword | Dakota Keyword Description |
| sqp | Uses a sequential quadratic programming method for underlying optimization |
| nip | Uses a nonlinear interior point method for underlying optimization |
| response_levels | Values at which to estimate desired statistics for each response |
| probability_levels | Specify probability levels at which to estimate the corresponding response value |
| gen_reliability_levels | Specify generalized reliability levels at which to estimate the corresponding response value |
| distribution | Selection of cumulative or complementary cumulative functions |
| Optional | model_pointer | Identifier for model block to be used by a method |

**Description**

Two local optimization methods are available: \texttt{sqp} (sequential quadratic programming or \texttt{nip} (nonlinear interior point method).

**Additional Resources**

See the topic page \texttt{evidence\_theory} for important background information and usage notes. Refer to \texttt{variable\_support} for information on supported variable types.

**See Also**

These keywords may also be of interest:

- \texttt{global\_evidence}
- \texttt{global\_interval\_est}
- \texttt{local\_interval\_est}

**sqp**

- \texttt{Keywords Area}
- \texttt{method}
- \texttt{local\_evidence}
- \texttt{sqp}

Uses a sequential quadratic programming method for underlying optimization

**Specification**

**Alias**: none

**Argument(s)**: none

**Description**

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The \texttt{sqp} keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.
6.2. METHOD

nip

- Keywords Area
- method
- local_evidence
- nip

Uses a nonlinear interior point method for underlying optimization

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The nip keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

**response_levels**

- Keywords Area
- method
- local_evidence
- response_levels

Values at which to estimate desired statistics for each response

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF probabilities/reliabilities to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_response_levels</td>
<td></td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
</tbody>
</table>
Optional  compute  Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

**Default Behavior**

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

**Expected Outputs**

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

**Usage Tips**

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

**Examples**

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```dakota
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```dakota
method,  
sampling,  
samples = 100 seed = 1  
complementary distribution  
response_levels = 3.6e+11 4.0e+11 4.4e+11  
6.0e+04 6.5e+04 7.0e+04  
3.5e+05 4.0e+05 4.5e+05

variables,  
normal_uncertain = 2  
means = 248.89, 593.33  
std_deviations = 12.4, 29.7  
descriptors = 'TF1n' 'TF2n'
```

6.2. METHOD

uniform_uncertain = 2
lower_bounds = 199.3, 474.63
upper_bounds = 298.5, 712.
descriptors = 'TF1u' 'TF2u'
weibull_uncertain = 2
alphas = 12., 30.
betas = 250., 590.
descriptors = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
num_pairs = 3, 4
abscissas = 5, 8, 10, .1, .2, .3, .4
counts = 17, 21, 0, 12, 24, 12, 0
descriptors = 'TF1h' 'TF2h'
histogram_point_uncertain
real = 1
num_pairs = 2
abscissas = 3, 4
counts = 1, 1
descriptors = 'TF3h'

interface,
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses,
response_functions = 3
no_gradients
no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.0341896485e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
<tr>
<td>3.5000000000e+05</td>
<td>4.0000000000e+05</td>
<td>8.6000000000e-06</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>4.5000000000e+05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e+01</td>
<td>5.5000000000e+01</td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e+01</td>
<td>3.8000000000e+01</td>
<td></td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e+01</td>
<td>2.3000000000e+01</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

num_response_levels

- Keywords Area
- method
- local_evidence
- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

Specification

Alias: none
Argument(s): INTEGERLIST
Default: response_levels evenly distributed among response functions

Description

The num_response_levels keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If num_response_levels is not specified, the response_levels will be evenly distributed among the responses.

Expected Outputs
6.2. METHOD

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

```
compute
  • Keywords Area
  • method
  • local_evidence
  • response_levels
  • compute
```

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen_reliabilities</td>
<td></td>
<td></td>
<td>Computes generalized reliabilities associated with response levels</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| Optional | system | Compute system reliability (series or parallel) |

**Description**

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

**Default Behavior**

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen-reliabilities`.

**Expected Output**

The type of statistics specified by `compute` will be reported for each response level.

**Usage Tips**

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

**Examples**

```method
sampling
  sample_type random
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
  6.0e+04 6.5e+04 7.0e+04
  3.5e+05 4.0e+05 4.5e+05
  compute reliabilities

probabilities
```

- **Keywords Area**
- **method**
- **local_evidence**
- **response_levels**
- **compute**
- **probabilities**

Computes probabilities associated with response levels

**Specification**

Alias: none

Argument(s): none
Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method sampling
  sample_type random
  samples = 100 seed = 1
complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
    6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
compute probabilities
```

gen_reliabilities

- Keywords Area
- method
- local_evidence
- response_levels
- compute
- gen_reliabilities

Computes generalized reliabilities associated with response levels

Specification

**Alias:** none

**Argument(s):** none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.
Examples

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
      6.0e+04 6.5e+04 7.0e+04
      3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

system

- **Keywords Area**
- **method**
- **local_evidence**
- **response_levels**
- **compute**
- **system**

Compute system reliability (series or parallel)

Specification

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>series</td>
<td>Aggregate response statistics assuming a series system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parallel</td>
<td>Aggregate response statistics assuming a parallel system</td>
</tr>
</tbody>
</table>

**Description**

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.
series
- Keywords Area
- method
- local_evidence
- response_levels
- compute
- system
- series

Aggregate response statistics assuming a series system

**Specification**

Alias: none
Argument(s): none

**Description**

See parent keyword system for description.

parallel
- Keywords Area
- method
- local_evidence
- response_levels
- compute
- system
- parallel

Aggregate response statistics assuming a parallel system

**Specification**

Alias: none
Argument(s): none

**Description**

See parent keyword system for description.
probability_levels

- Keywords Area
- method
- local_evidence
- probability_levels

Specify probability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>num_probability_levels</td>
<td>Specify which probability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

**Expected Output**

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

**Examples**

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
    samples = 100 seed = 1
  complementary distribution
  probability_levels = 1.66 .33 0.
    1.8 .5 0.
    1.3 .2 0.
variables,
  normal_uncertain = 2
  means = 248.89, 593.33
  std_deviations = 12.4, 29.7
```
6.2. METHOD

descriptors = 'TF1n' 'TF2n'
uniform_uncertain = 2
lower_bounds = 199.3, 474.63
upper_bounds = 298.5, 712.
descriptors = 'TF1u' 'TF2u'
weibull_uncertain = 2
alphas = 12., 30.
betas = 250., 590.
descriptors = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
num_pairs = 3 4
abscissas = 5 8 10 .1 .2 .3 .4
counts = 17 21 0 12 24 12 0
descriptors = 'TF1h' 'TF2h'
histogram_point_uncertain
real = 1
num_pairs = 2
abscissas = 3 4
counts = 1 1
descriptors = 'TF3h'

interface,
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses,
response_functions = 3
no_gradients
no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

| Probability Density Function (PDF) histograms for each response function: |
|:--------------------------:|:--------------------------:|:--------------------------:|
| **PDF for response_fn_1:** | **PDF for response_fn_2:** | **PDF for response_fn_3:** |
| **Bin Lower** | **Bin Lower** | **Bin Lower** |
| **Bin Upper** | **Bin Upper** | **Bin Upper** |
| **Density Value** | **Density Value** | **Density Value** |
| 2.7604749078e+11 | 3.4221494996e+11 | 5.1384774972e-12 |
| 3.4221494996e+11 | 4.0634975300e+11 | 5.1454122311e-12 |
| 4.0634975300e+11 | 5.4196114379e+11 | 2.4334239039e-12 |
| 4.6431154744e+04 | 5.651827775e+04 | 1.9839945149e-05 |
| 5.651827775e+04 | 6.1603813790e+04 | 5.8916108390e-05 |
| 6.1603813790e+04 | 7.8702465755e+04 | 2.9242071306e-05 |
| 2.3796737090e+05 | 3.6997214153e+05 | 5.3028386523e-06 |
| 3.6997214153e+05 | 3.8100966235e+05 | 9.0600055634e-06 |
| 3.8100966235e+05 | 4.811198127e+05 | 3.3274925348e-06 |

| Level mappings for each response function: |
| **Complementary Cumulative Distribution Function (CCDF) for response_fn_1:** |
| **Response Level** | **Probability Level** | **Reliability Index** | **General Rel Index** |
| **----------** | **----------** | **----------** | **----------** |
| 2.7604749078e+11 | 1.0000000000e+00 |  |  |
| 3.4221494996e+11 | 6.6000000000e-01 |  |  |
| 4.0634975300e+11 | 3.3000000000e-01 |  |  |
| 5.4196114379e+11 | 0.0000000000e+00 |  |  |
CHAPTER 6. KEYWORDS AREA

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>5.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.8702465755e+04</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>2.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4111498127e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_probability_levels

- Keywords Area
- method
- local_evidence
- probability_levels
- num_probability_levels

Specify which probability_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page
**6.2. METHOD**

```plaintext
gen_reliability_levels
  
  Keywords Area

  method

  local_evidence

  gen_reliability_levels

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias**: none

**Argument(s)**: REALLIST

**Default**: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>num_gen_reliability_levels</td>
<td>Specify which gen_reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

```plaintext
num_gen_reliability_levels
  
  Keywords Area

  method

  local_evidence

  gen_reliability_levels

  num_gen_reliability_levels

Specify which gen_reliability_levels correspond to which response
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none
Default: gen_reliability.levels evenly distributed among response functions

Description

See parent page

distribution

- Keywords Area
- method
- local_evidence
- distribution

Selection of cumulative or complementary cumulative functions

Specification

Alias: none
Argument(s): none
Default: cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword cumulative</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Computes statistics according to cumulative functions</td>
</tr>
</tbody>
</table>

complementary

Computes statistics according to complementary cumulative functions

Description

The distribution keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the distribution keyword is present, it must be accompanied by either cumulative or complementary. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.
6.2. METHOD

Examples

```plaintext
method
  sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```

cumulative

- **Keywords Area**
- **method**
- **local_evidence**
- **distribution**
- **cumulative**

Computes statistics according to cumulative functions

Specification

**Alias:** none

**Argument(s):** none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```plaintext
method
  sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```

cumulative

- **Keywords Area**
- **method**
- **local_evidence**
- **distribution**
• complementary

Computes statistics according to complementary cumulative functions

**Specification**

Alias: none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

**model_pointer**

• **Keywords Area**

• **method**

• **local_evidence**

• **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

• **block_pointer**

**Specification**

Alias: none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model-pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

```
environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURRE'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURRE'
  surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluationConcurrency = 5
  analysis_driver = 'text_book'

responses
```
6.2.62  local_interval_est

- **Keywords Area**
- **method**
- **local_interval_est**

Interval analysis using local optimization

**Topics**

This keyword is related to the topics:

- **uncertainty_quantification**
- **epistemic_uncertainty_quantification_methods**
- **interval_estimation**

**Specification**

**Alias:** nond_local_interval_est
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>spq</td>
<td>sqp</td>
<td>Uses a sequential quadratic programming method for underlying optimization</td>
</tr>
<tr>
<td>nip</td>
<td></td>
<td></td>
<td>Uses a nonlinear interior point method for underlying optimization</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>convergence_tolerance</th>
<th>Stopping criterion based on convergence of the objective function or statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

**Description**

Interval analysis using local methods (\texttt{local_interval_est}). If the problem is amenable to local optimization methods (e.g. can provide derivatives or use finite difference method to calculate derivatives), then one can use one of two local methods to calculate these bounds.

- \texttt{sqp}
- \texttt{nip}

**Additional Resources**

Refer to \texttt{variable_support} for information on supported variable types.

**Theory**

In interval analysis, one assumes that nothing is known about an epistemic uncertain variable except that its value lies somewhere within an interval. In this situation, it is NOT assumed that the value has a uniform probability of occurring within the interval. Instead, the interpretation is that any value within the interval is a possible value or a potential realization of that variable. In interval analysis, the uncertainty quantification problem is one of determining the resulting bounds on the output (defining the output interval) given interval bounds on the inputs. Again, any output response that falls within the output interval is a possible output with no frequency information assigned to it.

**See Also**

These keywords may also be of interest:

- \texttt{global_evidence}
- \texttt{global_interval_est}
- \texttt{local_evidence}

**sqp**

- Keywords Area
- method
- \texttt{local_interval_est}
- \texttt{sqp}

Uses a sequential quadratic programming method for underlying optimization.
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
Argument(s): none

Description
Many uncertainty quantification methods solve a constrained optimization problem under the hood. The sqp keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

nip
- Keywords Area
- method
- local_interval_est
- nip

Uses a nonlinear interior point method for underlying optimization

Specification
Alias: none
Argument(s): none

Description
Many uncertainty quantification methods solve a constrained optimization problem under the hood. The nip keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

convergence_tolerance
- Keywords Area
- method
- local_interval_est
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics
This keyword is related to the topics:
- method_independent_controls
Specification

**Alias:** none
**Argument(s):** REAL
**Default:** 1.e-4

**Description**

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

**Behavior Varies by Package/Library**

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- **DOT:** must be satisfied for two consecutive iterations
- **NPSOL:** defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- **NL2SOL:** See nl2sol

**model_pointer**

- **Keywords Area**
- **method**
- **local_interval_est**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
  tabular_graphics_data
  method_pointer = ‘UQ’

method
  id_method = ‘UQ’
  model_pointer = ‘SURR’
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = ‘SURR’
  surrogate global,
  dace_method_pointer = ‘DACE’
  polynomial quadratic

method
  id_method = ‘DACE’
  model_pointer = ‘DACE_M’
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = ‘DACE_M’
  single
  interface_pointer = ’I1’

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = ‘x1’ ‘x2’
6.2. METHOD

interface
  id_interface = 'II'
system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.63 local_reliability

- Keywords Area
- method
- local_reliability

Local reliability method

Topics
This keyword is related to the topics:
- uncertainty_quantification
- reliability_methods

Specification

Alias: nond_local_reliability
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>mpp_search</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specify which</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MPP search option</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>to use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>response_levels</td>
<td>Values at which to estimate desired</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>statistics for each response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reliability_levels</td>
<td>Specify reliability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>levels at which the response values</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>will be estimated</td>
</tr>
</tbody>
</table>
### Optional Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>max_iterations</code></td>
<td>Stopping criterion based on number of iterations</td>
</tr>
<tr>
<td><code>convergence_tolerance</code></td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td><code>distribution</code></td>
<td>Selection of cumulative or complementary cumulative functions</td>
</tr>
<tr>
<td><code>probability_levels</code></td>
<td>Specify probability levels at which to estimate the corresponding response value</td>
</tr>
<tr>
<td><code>gen_reliability_levels</code></td>
<td>Specify generalized reliability levels at which to estimate the corresponding response value</td>
</tr>
<tr>
<td><code>model_pointer</code></td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

### Description

Local reliability methods compute approximate response function distribution statistics based on specified uncertain variable probability distributions. Each of the local reliability methods can compute forward and inverse mappings involving response, probability, reliability, and generalized reliability levels.

The forward reliability analysis algorithm of computing reliabilities/probabilities for specified response levels is called the Reliability Index Approach (RIA), and the inverse reliability analysis algorithm of computing response levels for specified probability levels is called the Performance Measure Approach (PMA).

The different RIA/PMA algorithm options are specified using the `mpp_search` specification which selects among different limit state approximations that can be used to reduce computational expense during the MPP searches.

### Theory

The Mean Value method (MV, also known as MVFOSM in [42]) is the simplest, least-expensive method in that it estimates the response means, response standard deviations, and all CDF/CCDF forward/inverse mappings from a single evaluation of response functions and gradients at the uncertain variable means. This approximation can
have acceptable accuracy when the response functions are nearly linear and their distributions are approximately Gaussian, but can have poor accuracy in other situations.

All other reliability methods perform an internal nonlinear optimization to compute a most probable point (M-PP) of failure. A sign convention and the distance of the MPP from the origin in the transformed standard normal space ("u-space") define the reliability index, as explained in the section on Reliability Methods in the Uncertainty Quantification chapter of the Users Manual [4]. Also refer to variable support for additional information on supported variable types for transformations to standard normal space. The reliability can then be converted to a probability using either first- or second-order integration, may then be refined using importance sampling, and finally may be converted to a generalized reliability index.

See Also
These keywords may also be of interest:

- adaptive_sampling
- gpais
- global_reliability
- sampling
- importance_sampling
- polynomial_chaos
- stoch_collocation

mpp_search

- Keywords Area
- method
- local_reliability
- mpp_search

Specify which MPP search option to use

Topics
This keyword is related to the topics:

- uncertainty_quantification
- reliability_methods

Specification

Alias: none

Argument(s): none

Default: No MPP search (MV method)
<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required <em>(Choose One)</em></td>
<td>Group 1</td>
<td>x_taylor_mean</td>
<td>Form Taylor series approximation in &quot;x-space&quot; at variable means</td>
</tr>
<tr>
<td></td>
<td></td>
<td>u_taylor_mean</td>
<td>Form Taylor series approximation in &quot;u-space&quot; at variable means</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x_taylor_mpp</td>
<td>X-space Taylor series approximation with iterative updates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>u_taylor_mpp</td>
<td>U-space Taylor series approximation with iterative updates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x_two_point</td>
<td>Predict MPP using Two-point Adaptive Nonlinear Approximation in &quot;x-space&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>u_two_point</td>
<td>Predict MPP using Two-point Adaptive Nonlinear Approximation in &quot;u-space&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_approx</td>
<td>Perform MPP search on original response functions (use no approximation)</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional (Choose One)</th>
<th>Group 2</th>
<th>sqp</th>
<th>Uses a sequential quadratic programming method for underlying optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>nip</td>
<td>Uses a nonlinear interior point method for underlying optimization</td>
</tr>
</tbody>
</table>

Optional integration

Description

The x\_taylor\_mean MPP search option performs a single Taylor series approximation in the space of the original uncertain variables ("x-space") centered at the uncertain variable means, searches for the MPP for each response/probability level using this approximation, and performs a validation response evaluation at each predicted MPP. This option is commonly known as the Advanced Mean Value (AMV) method. The u\_taylor\_mean option is identical to the x\_taylor\_mean option, except that the approximation is performed in u-space. The x\_taylor\_mpp approach starts with an x-space Taylor series at the uncertain variable means, but iteratively updates the Taylor series approximation at each MPP prediction until the MPP converges. This option is commonly known as the AMV+ method. The u\_taylor\_mpp option is identical to the x\_taylor\_mpp option, except that all approximations are performed in u-space. The order of the Taylor-series approximation is determined by the corresponding responses specification and may be first or second-order. If second-order (methods named AMV\^2 and AMV\^2+ in [22]), the series may employ analytic, finite difference, or quasi Hessians (BFGS or SR1). The x\_two\_point MPP search option uses an x-space Taylor series approximation at the uncertain variable means for the initial MPP prediction, then utilizes the Two-point Adaptive Nonlinear Approximation (TANA) outlined in [91] for all subsequent MPP predictions. The u\_two\_point approach is identical to x\_two\_point, but all the approximations are performed in u-space. The x\_taylor\_mpp and u\_taylor\_mpp, x\_two\_point and u\_two\_point approaches utilize the max_iterations and convergence_tolerance method independent controls to control the convergence of the MPP iterations (the maximum number of MPP iterations per level is limited by max_iterations, and the MPP iterations are considered converged when \( \| u^{k+1} - u^k \|_2 < \text{convergence_tolerance} \)). And, finally, the no_approx option performs the MPP search on the original response functions without the use of any approximations. The optimization algorithm used to perform these MPP searches can be selected to be either sequential quadratic programming (uses the npsol.sqp optimizer) or nonlinear interior point (uses the optpp.q.newton optimizer) algorithms using the sqp or nip keywords.

In addition to the MPP search specifications, one may select among different integration approaches for computing probabilities at the MPP by using the integration keyword followed by either first_order or second_order. Second-order integration employs the formulation of [50] (the approach of [13] and the correction of [51] are also implemented, but are not active). Combining the no_approx option of the MPP search with first- and second-order integrations results in the traditional first- and second-order reliability methods (FORM and SORM). These integration approximations may be subsequently refined using importance sampling. The refinement specification allows the selection of basic importance sampling (import), adaptive importance sampling (adapt_import), or multimodal adaptive importance sampling (mm_adapt_import), along with the specification of number of samples (samples) and random seed (seed). Additional details on these methods are available in [24] and [22] and in the Uncertainty Quantification Capabilities chapter of the Users Manual.
CHAPTER 6. KEYWORDS AREA

x_taylor_mean

- Keywords Area
- method
- local_reliability
- mpp_search
- x_taylor_mean

Form Taylor series approximation in "x-space" at variable means

Topics

This keyword is related to the topics:

- reliability_methods

Specification

Alias: none

Argument(s): none

Description

This mpp_search option performs a single Taylor series approximation in the space of the original uncertain variables ("x-space") centered at the uncertain variable means, searches for the MPP for each response/probability level using this approximation, and performs a validation response evaluation at each predicted MPP. This option is commonly known as the Advanced Mean Value (AMV) method.

u_taylor_mean

- Keywords Area
- method
- local_reliability
- mpp_search
- u_taylor_mean

Form Taylor series approximation in "u-space" at variable means

Topics

This keyword is related to the topics:

- reliability_methods
6.2. METHOD

Specification
Alias: none
  Argument(s): none

Description
This mpp_search option performs a single Taylor series approximation in the transformed space of the uncertain variables (“u-space”) centered at the uncertain variable means. This option is commonly known as the Advanced Mean Value (AMV) method, but is performed in u-space instead of x-space.

x_taylor_mpp
  • Keywords Area
  • method
  • local_reliability
  • mpp_search
  • x_taylor_mpp
    X-space Taylor series approximation with iterative updates

Topics
This keyword is related to the topics:
  • reliability_methods

Specification
Alias: none
  Argument(s): none

Description
This mpp_search option starts with an x-space Taylor series at the uncertain variable means, but iteratively updates the Taylor series approximation at each MPP prediction until the MPP converges. This option is commonly known as the AMV+ method.

u_taylor_mpp
  • Keywords Area
  • method
  • local_reliability
  • mpp_search
  • u_taylor_mpp
    U-space Taylor series approximation with iterative updates
Topics

This keyword is related to the topics:

- reliability_methods

Specification

Alias: none

Argument(s): none

Description

This mpp_search option starts with a u-space Taylor series at the uncertain variable means, and iteratively updates the Taylor series approximation at each MPP prediction until the MPP converges. This option is commonly known as the AMV+ method and is identify to x_taylor_mpp except that it is performed in u-space.

x_two_point

- Keywords Area
- method
- local_reliability
- mpp_search
- x_two_point

Predict MPP using Two-point Adaptive Nonlinear Approximation in “x-space”

Topics

This keyword is related to the topics:

- reliability_methods

Specification

Alias: none

Argument(s): none

Description

This mpp_search option uses an x-space Taylor series approximation at the uncertain variable means for the initial MPP prediction, then utilizes the Two-point Adaptive Nonlinear Approximation (TANA) outlined in [ Xu98 "Xu and Grandhi, 1998"] for all subsequent MPP predictions.
6.2. **METHOD**

**u_two_point**

- **Keywords Area**
- method
- local_reliability
- mpp_search
- u_two_point

Predict MPP using Two-point Adaptive Nonlinear Approximation in "u-space"

**Topics**

This keyword is related to the topics:

- reliability_methods

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

This mpp_search option is identical to `x_two_point`, but it performs the Two-point Adaptive Nonlinear Approximation (TANA) in u-space instead of x-space.

**no_approx**

- **Keywords Area**
- method
- local_reliability
- mpp_search
- no_approx

Perform MPP search on original response functions (use no approximation)

**Topics**

This keyword is related to the topics:

- reliability_methods

**Specification**

**Alias:** none

**Argument(s):** none
Description

This mpp_search option performs the MPP search on the original response functions without the use of any approximations. Note that the use of the no_approx MPP search with first-order probability integration results in the traditional reliability method called FORM (First-Order Reliability Method). Similarly, the use of no_approx with second-order probability integration results in SORM (Second-Order Reliability Method).

sqp

- Keywords Area
- method
- local_reliability
- mpp_search
- sqp

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none
Argument(s): none
Default: sqp

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The sqp keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

nip

- Keywords Area
- method
- local_reliability
- mpp_search
- nip

Uses a nonlinear interior point method for underlying optimization

Specification

Alias: none
Argument(s): none
Default: sqp
6.2. **METHOD**

**Description**

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

**integration**

- **Keywords Area**
- **method**
- **local_reliability**
- **mpp_search**
- **integration**

Integration approach

**Topics**

This keyword is related to the topics:

- **reliability_methods**

**Specification**

*Alias:* none

*Argument(s):* none

*Default:* First-order integration

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword first_order</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>first_order</td>
<td>First-order integration scheme</td>
</tr>
<tr>
<td></td>
<td></td>
<td>second_order</td>
<td>Second-order integration scheme</td>
</tr>
<tr>
<td>Optional</td>
<td>probability_refinement</td>
<td></td>
<td>Allow refinement of probability and generalized reliability results using importance sampling</td>
</tr>
</tbody>
</table>

**Description**

This keyword controls how the probabilities at the MPP are computed: integration is followed by either `first_order` or `second_order`, indicating the order of the probability integration.
first_order

- Keywords Area
- method
- local_reliability
- mpp_search
- integration
- first_order

First-order integration scheme

Topics

This keyword is related to the topics:

- reliability_methods

Specification

Alias: none
Argument(s): none

Description

First-order integration in local reliability methods uses the minimum Euclidean distance from the origin to the most probable point (MPP) in transformed space to compute the probability of failure. This distance, commonly called the reliability index Beta, is used to calculate the probability of failure by calculating the standard normal cumulative distribution function at -Beta.

second_order

- Keywords Area
- method
- local_reliability
- mpp_search
- integration
- second_order

Second-order integration scheme

Topics

This keyword is related to the topics:

- reliability_methods
6.2. METHOD

Specification

Alias: none

Argument(s): none

Description

Second-order integration in local reliability methods modifies the first-order integration approach to apply a curvature correction. This correction is based on the formulation of [Hoh88 “Hohenbichler and Rackwitz, 1988”].

```
probability_refinement
```

- Keywords Area
- method
- local_reliability
- mpp_search
- integration
- probability_refinement

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- reliability_methods

Specification

Alias: sample_refinement

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>import</td>
<td>Sampling option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adapt_import</td>
<td>Importance sampling option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mm_adapt_import</td>
<td>Sampling option</td>
</tr>
<tr>
<td>Optional</td>
<td>refinement_-samples</td>
<td>Specify the number of samples used to improve a probability estimate.</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| Optional | seed | Seed of the random number generator |

**Description**

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

**import**

- Keywords Area
- method
- local_reliability
- mpp_search
- integration
- probability_refinement
- import

Sampling option

**Specification**

Alias: none

**Argument(s):** none

**Description**

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

**adapt_import**

- Keywords Area
- method
- local_reliability
- mpp_search
- integration
6.2. METHOD

- probability_refinement
- adapt_import

Importance sampling option

Specification
Alias: none
Argument(s): none

Description
adapt_import centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

mm_adapt_import
- Keywords Area
- method
- local_reliability
- mpp_search
- integration
- probability_refinement
- mm_adapt_import

Sampling option

Specification
Alias: none
Argument(s): none

Description
mm_adapt_import starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then mm_adapt_import proceeds similarly to adapt_import (sample until convergence).
refinement_samples

- Keywords Area
- method
- local_reliability
- mpp_search
- integration
- probability_refinement
- refinement_samples

Specify the number of samples used to improve a probability estimate.

**Specification**

Alias: none  
**Argument(s):** INTEGER

**Description**

Specify the number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

seed

- Keywords Area
- method
- local_reliability
- mpp_search
- integration
- probability_refinement
- seed

Seed of the random number generator

**Specification**

Alias: none  
**Argument(s):** INTEGER  
**Default:** system-generated (non-repeatable)
6.2. METHOD

Description
The random seed control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior
If not specified, the seed is randomly generated.

Expected Output
If seed is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips
If a stochastic study was run without seed specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

response_levels

- Keywords Area
- method
- local_reliability
- response_levels

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_response_levels</td>
<td>num_response_levels</td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>compute</th>
<th>Selection of statistics to compute at each response level</th>
</tr>
</thead>
</table>

Description

The response_levels specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

**Default Behavior**

If response_levels are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

**Expected Outputs**

If response_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

**Usage Tips**

The num_response_levels is used to specify which arguments of the response_level correspond to which response.

**Examples**

For example, specifying a response_level of 52.3 followed with compute probabilities will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the num_response_levels key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
            6.0e+04 6.5e+04 7.0e+04
            3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means = 248.89, 593.33
  std_deviations = 12.4, 29.7
  descriptors = 'TF1n' 'TF2n'
```
uniform_uncertain = 2
lower_bounds = 199.3, 474.63
upper_bounds = 298.5, 712.
descriptors = 'TF1u', 'TF2u'

weibull_uncertain = 2
alphas = 12., 30.
betas = 250., 590.
descriptors = 'TF1w', 'TF2w'

histogram_bin_uncertain = 2
num_pairs = 3, 4
abscissas = 5, 8, 10, .1, .2, .3, .4
counts = 17, 21, 0, 12, 24, 12, 0
descriptors = 'TF1h', 'TF2h'

histogram_point_uncertain
real = 1
num_pairs = 2
abscissas = 3, 4
counts = 1, 1
descriptors = 'TF3h'

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.4196114379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702466755e+04</td>
<td>1.0341896485e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:
<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
<tr>
<td>3.5000000000e+05</td>
<td>4.0000000000e+05</td>
<td>8.6000000000e-06</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>4.5000000000e+05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+01</td>
<td>5.5000000000e+00</td>
<td>2.8742313192e-05</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>4.0000000000e+01</td>
<td>3.8000000000e+00</td>
<td>2.2557612778e-12</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>4.4000000000e+01</td>
<td>2.3000000000e+00</td>
<td>1.0341896485e-05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

**num_response_levels**

- Keywords Area
- method
- local_reliability
- response_levels
- num_response_levels

Number of values at which to estimate desired statistics for each response

**Specification**

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

**Description**

The num_response_levels keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

**Default Behavior**

If num_response_levels is not specified, the response_levels will be evenly distributed among the responses.

**Expected Outputs**
The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```plaintext
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5

compute
  • Keywords Area
  • method
  • local_reliability
  • response_levels
  • compute

Selection of statistics to compute at each response level

Specification

Alias: none
Argument(s): none
Default: probabilities

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>probabilities</td>
<td>Computes probabilities associated with response levels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reliabilities</td>
<td>Computes reliabilities associated with response levels</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>system</th>
<th>gen_reliabilities</th>
<th>Computes generalized reliabilities associated with response levels</th>
</tr>
</thead>
</table>

**Description**

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

**Default Behavior**

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

**Expected Output**

The type of statistics specified by `compute` will be reported for each response level.

**Usage Tips**

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach. CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

**Examples**

```
method
sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
    6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

**probabilities**

- `Keywords Area`
- `method`
- `local_reliability`
- `response_levels`
- `compute`
- `probabilities`

Computes probabilities associated with response levels
6.2. METHOD

Specification

Alias: none

Argument(s): none

Description

The probabilities keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If response_levels is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the probabilities keyword should be specified in conjunction with the compute keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                  6.0e+04 6.5e+04 7.0e+04
                  3.5e+05 4.0e+05 4.5e+05
  compute probabilities

reliabilities
```

- Keywords Area
- method
- local_reliability
- response_levels
- compute
- reliabilities

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```method
sampling
  sample_type random
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
  6.0e+04 6.5e+04 7.0e+04
  3.5e+05 4.0e+05 4.5e+05
  compute reliabilities

  gen_reliabilities
  
  • Keywords Area
  
  • method
  
  • local_reliability
  
  • response_levels
  
  • compute
  
  • gen_reliabilities

Computes generalized reliabilities associated with response levels
```

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.
Examples

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

Computes system reliability (series or parallel)

Specification

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group 1</td>
<td>series</td>
<td>Description</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td></td>
<td>Aggregate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>response statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>assuming a series</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parallel</td>
<td>Aggregate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>response statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>assuming a parallel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>system</td>
</tr>
</tbody>
</table>

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.
series

- Keywords Area
- method
- local_reliability
- response_levels
- compute
- system
- series

Aggregate response statistics assuming a series system

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

See parent keyword system for description.

parallel

- Keywords Area
- method
- local_reliability
- response_levels
- compute
- system
- parallel

Aggregate response statistics assuming a parallel system

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

See parent keyword system for description.
6.2. METHOD

reliability_levels

- Keywords Area
- method
- local_reliability
- reliability_levels

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

| Required/- | Description of | Dakota Keyword | Dakota Keyword | Description |
| Optional | Group | | num_reliability_levels | Specify which reliability_levels correspond to which response |

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

num_reliability_levels

- Keywords Area
- method
- local_reliability
- reliability_levels
- num_reliability_levels

Specify which reliability_levels correspond to which response
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGERLIST

Default: reliability_levels evenly distributed among response functions

Description

See parent page

max_iterations

- Keywords Area
- method
- local_reliability
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25∗n)

Description

The maximum number of iterations (default: 100). See also max_function_evaluations.

convergence_tolerance

- Keywords Area
- method
- local_reliability
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics
6.2. METHOD

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

distribution

- Keywords Area
- method
- local_reliability
- distribution

Selection of cumulative or complementary cumulative functions
Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)
### Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

**Default Behavior**

- If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

**Expected Outputs**

- Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

### Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

**cumulative**

- **Keywords Area**
- **method**
- **local_reliability**
- **distribution**
- **cumulative**

  Computes statistics according to cumulative functions

### Specification

**Alias:** none

**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description
Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior
By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the cumulative keyword must be appear in conjunction with the distribution keyword.

Expected Outputs
Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

method sampling
  sample_type lhs
  samples = 10
  distribution cumulative

complementary
  • Keywords Area
  • method
  • local_reliability
  • distribution
  • complementary

Computes statistics according to complementary cumulative functions

Specification
Alias: none
Argument(s): none

Description
Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior
By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must be appear in conjunction with the distribution keyword.

Expected Outputs
Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.
6.2. METHOD

Examples

```
method sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

```
probability_levels

- Keywords Area
- method
- local_reliability
- probability_levels

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_probability_levels</td>
<td></td>
<td>Specify which probability_levels correspond to which response</td>
</tr>
</tbody>
</table>

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If probability_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
    sampling,
    samples = 100 seed = 1
    complementary distribution
```
probability_levels = 1.  .66  .33  0.
    1.  .8  .5  0.
    1.  .3  .2  0.

variables,
    normal_uncertain = 2
        means = 248.89, 593.33  
        std_deviations = 12.4, 29.7  
        descriptors = 'TF1n' 'TF2n'
    uniform_uncertain = 2
        lower_bounds = 199.3, 474.63  
        upper_bounds = 298.5, 712.  
        descriptors = 'TF1u' 'TF2u'
    weibull_uncertain = 2
        alphas = 12., 30.  
        betas = 250., 590.  
        descriptors = 'TF1w' 'TF2w'
    histogram_bin_uncertain = 2
        num_pairs = 3  4  
        abscissas = 5  8 10 .1 .2 .3 .4  
        counts = 17 21 0 12 24 12 0  
        descriptors = 'TF1h' 'TF2h'
    histogram_point_uncertain
        real = 1  
        num_pairs = 2  
        abscissas = 3  4  
        counts = 1 1  
        descriptors = 'TF3h'

interface,
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses,
    response_functions = 3
    no_gradients
    no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0634975300e+11</td>
<td>5.1454122311e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.4196114379e+11</td>
<td>2.4334239039e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.6997214153e+05</td>
<td>5.3028386523e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.8100966235e+05</td>
<td>9.0600055634e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.4111498127e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>
6.2. METHOD

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.6000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>3.3000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4196114379e+11</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6431154744e+04</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>5.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.8702465755e+04</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>1.0000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.699721453e+05</td>
<td>3.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>2.0000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4111498127e+05</td>
<td>0.0000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_probability_levels**

- **Keywords** Area
- **method**
- **local_reliability**
- **probability_levels**
- **num_probability_levels**

Specify which probability_levels correspond to which response

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** probability_levels evenly distributed among response functions

**Description**

See parent page
CHAPTER 6. KEYWORDS AREA

**gen_reliability_levels**

- **Keywords Area**
- **method**
- **local_reliability**
- **gen_reliability_levels**

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_gen_reliability_levels</td>
<td></td>
<td>Specify which gen_reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

**Theory**

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_gen_reliability_levels**

- **Keywords Area**
- **method**
- **local_reliability**
- **gen_reliability_levels**
- **num_gen_reliability_levels**

Specify which gen_reliability_levels correspond to which response
6.2. METHOD

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability_levels evenly distributed among response functions

Description

See parent page

model_pointer

- Keywords Area
- method
- local_reliability
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model-pointer for each method is imperative.

See block_pointer for details about pointers.
Examples

environment
  tabular_graphics_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
    0.1 0.2 0.6
    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
    lower_bounds = 0. 0.
    upper_bounds = 1. 1.
    descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.64 global_reliability

- Keywords Area
- method
- global_reliability

Global reliability methods
6.2. METHOD

Topics
This keyword is related to the topics:

- uncertainty_quantification
- reliability_methods

Specification
Alias: nond_global_reliability
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>x_gaussian_process</td>
<td>u_gaussian_process</td>
<td>Create GP surrogate in x-space</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>Group 2</td>
<td>surfpack</td>
<td>Use the Surfpack version of Gaussian Process surrogates</td>
</tr>
<tr>
<td>Optional</td>
<td>dakota</td>
<td>Select the built in Gaussian Process surrogate</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>import_build-points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>export_approx-points_file</td>
<td>Output file for evaluations of a surrogate model</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>seed</td>
<td>Seed of the random number generator</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>rng</td>
<td>Selection of a random number generator</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>response_levels</td>
<td>Values at which to estimate desired statistics for each response</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>distribution</td>
<td>Selection of cumulative or complementary cumulative functions</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>probability_levels</td>
<td>Specify probability levels at which to estimate the corresponding response value</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>gen_reliability_levels</td>
<td>Specify generalized reliability levels at which to estimate the corresponding response value</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

These methods do not support forward/inverse mappings involving `reliability_levels`, since they never form a reliability index based on distance in u-space. Rather they use a Gaussian process model to form an approximation to the limit state (based either in x-space via the `x_gaussian_process` specification or in u-space via the `u_gaussian_process` specification), followed by probability estimation based on multimodal adaptive importance sampling (see [11]) and [12]). These probability estimates may then be transformed into generalized reliability levels if desired. At this time, inverse reliability analysis (mapping probability or generalized reliability levels into response levels) is not implemented.
The Gaussian process model approximation to the limit state is formed over the aleatory uncertain variables by default, but may be extended to also capture the effect of design, epistemic uncertain, and state variables. If this is desired, one must use the appropriate controls to specify the active variables in the variables specification block. Refer to variable_support for additional information on supported variable types.

See Also
These keywords may also be of interest:
- adaptive_sampling
- gpais
- local_reliability
- sampling
- importance_sampling
- polynomial_chaos
- stoch_collocation

x_gaussian_process
- Keywords Area
- method
- global_reliability
- x_gaussian_process

Create GP surrogate in x-space

Topics
This keyword is related to the topics:
- reliability_methods

Specification
Alias: x_kriging
Argument(s): none

u_gaussian_process
- Keywords Area
- method
- global_reliability
- u_gaussian_process

Create GP surrogate in u-space
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:
- reliability_Methods

Specification
Alias: u_kriging
Argument(s): none

surfpack
- Keywords Area
- method
- global_reliability
- surfpack

Use the Surfpack version of Gaussian Process surrogates

Description
This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:
   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See optimization_method.

   The total number of evaluations of the likelihood function can be controlled using the max_trials keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:
   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See trend.

3. Correlation Lengths:
   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See correlation_lengths.
4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The surfpack model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The use_derivatives keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here “inexpensive” means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

dakota

- Keywords Area
- method
- global_reliability
- dakota

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none
CHAPTER 6. KEYWORDS AREA

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the **gaussian process** dakota model can be found in[58].

Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a “nugget,” but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a **point_selection** option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the **point_selection** option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

**import_build_points_file**

- **Keywords Area**
- **method**
- **global_reliability**
- **import_build_points_file**

File containing points you wish to use to build a surrogate

Specification

**Alias:** import_points_file  
**Argument(s):** STRING  
**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Optional/Choose One</th>
<th>Description of Group tabular format (Group 1)</th>
<th>Dakota Keyword annotated</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>custom.annotated</td>
</tr>
</tbody>
</table>

Sets annotated tabular file format
6.2. METHOD

<table>
<thead>
<tr>
<th>freeform</th>
<th>Selects freeform file format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>active_only</td>
</tr>
<tr>
<td></td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- annotated (default)
- custom.annotated
- freeform

**Examples**

```plaintext
method
  polynomial_chaos
  expansion_order = 4
  import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

**annotated**

- **Keywords Area**
- **method**
- **global.reliability**
- **import_build_points_file**
- **annotated**

  Selects annotated tabular file format

**Topics**

This keyword is related to the topics:

- **file_formats**
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

• To specify pre-Dakota 6.1 tabular format, which did not include interface id, specify custom_annotated header eval_id

• Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

• When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

• Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.9009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

custom_annotated

• Keywords Area

• method

• global_reliability

• import_build_points_file

• custom_annotated

Selects custom-annotated tabular file format
6.2. METHOD

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description
A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior
The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id

Resulting tabular file:

%eval_id  x1      x2     obj_fn  nln_ineq_con_1  nln_ineq_con_2
1    0.9      1.1    0.0002   0.26       0.76
2  0.90009   1.1  0.0001996404857  0.2601620081   0.759955
3  0.89991   1.1  0.0002003604863  0.2598380081   0.760045
...

header
  • Keywords Area
  • method
  • global_reliability
  • import_build_points_file
  • custom_annotated
  • header

Enable header row in custom-annotated tabular file

Specification

Alias: none
  Argument(s): none
  Default: no header

Description

See description of parent custom_annotated

eval_id
  • Keywords Area
  • method
  • global_reliability
  • import_build_points_file
  • custom_annotated
  • eval_id

Enable evaluation ID column in custom-annotated tabular file
6.2. METHOD

Specification
Alias: none
Argument(s): none
Default: no eval_id column

Description
See description of parent custom_annotated

interface_id
- Keywords Area
- method
- global_reliability
- import_build_points_file
- custom_annotated
- interface_id
Enable interface ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no interface_id column

Description
See description of parent custom_annotated

freeform
- Keywords Area
- method
- global_reliability
- import_build_points_file
- freeform
Selects freeform file format

Topics
This keyword is related to the topics:
- file_formats
### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** annotated format

### Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The annotated format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

### Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
Resulting tabular file:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
</tbody>
</table>
```

**active_only**

- **Keywords Area**
- **method**
- **global_reliability**
- **import_build_points_file**
- **active_only**

Import only active variables from tabular data file
6.2. METHOD

Topics

This keyword is related to the topics:

- file formats

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword active\_only indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

**export\_approx\_points\_file**

- **Keywords Area**
  - method
  - global\_reliability
  - export\_approx\_points\_file

Output file for evaluations of a surrogate model

**Specification**

Alias: export\_points\_file

Argument(s): STRING

Default: no point export to a file

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

| freeform | Selects freeform file format |

Description

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- annotated (default)
- custom_annotated
- freeform

annotated

- Keywords Area
- method
- global_reliability
- export_approx_points_file
- annotated

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips
6.2. METHOD

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated
  header eval_id

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to
  annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but
  extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns,
and data for variables and responses. Input file fragment:

```
environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated

Resulting tabular file:
```

<table>
<thead>
<tr>
<th>eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.9009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
</tbody>
</table>

```
custom_annotated

- Keywords Area
- method
- global_reliability
- export_approx_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
Argument(s): none
Default: annotated format
### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

#### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom.annotated, followed by options, in the relevant export/import context.

#### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

#### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```plaintext
%eval_id   x1   x2     obj_fn nln_ineq_con_1 nln_ineq_con_2
1   0.9   1.1    0.0002  0.26   0.76
2   0.90009 1.1  0.0001996404857  0.2601620081  0.759955
3   0.89991 1.1  0.0002003604863  0.2598380081  0.760045
...```
6.2. METHOD

header

- Keywords Area
- method
- global_reliability
- export_approx_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no header

**Description**

See description of parent **custom_annotated**

**eval_id**

- Keywords Area
- method
- global_reliability
- export_approx_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no eval_id column

**Description**

See description of parent **custom_annotated**
interface_id

- Keywords Area
- method
- global_reliability
- export_approx_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none

Argument(s): none

Default: no interface_id column

**Description**

See description of parent custom_annotated

**freeform**

- Keywords Area
- method
- global_reliability
- export_approx_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: none

Argument(s): none

Default: annotated format
6.2. METHOD

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
    tabular_data
        tabular_data_file = 'dakota_summary.dat'
    freeform
Resulting tabular file:

    0.9   1.1   0.0002  0.26   0.76
    0.90009 1.1 0.0001996404857 0.2601620081 0.759955
    0.89991 1.1 0.0002003604863 0.2598380081 0.760045
    ...
```

use_derivatives

- Keywords Area
- method
- global_reliability
- use_derivatives

Use derivative data to construct surrogate models

Specification
Alias: none

Argument(s): none
Default: use function values only
CHAPTER 6. KEYWORDS AREA

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

**seed**

- Keywords Area
- method
- global_reliability
- seed

Seed of the random number generator

Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

**Default Behavior**

If not specified, the seed is randomly generated.

**Expected Output**

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

**Usage Tips**

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

**rng**

- Keywords Area
- method
- global_reliability
6.2. METHOD

- **rng**
  
  Selection of a random number generator

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** Mersenne twister (mt19937)

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mt19937</strong></td>
<td></td>
<td></td>
<td>Generates random numbers using the Mersenne twister</td>
</tr>
<tr>
<td><strong>rnum2</strong></td>
<td></td>
<td></td>
<td>Generates pseudo-random numbers using the Pecos package</td>
</tr>
</tbody>
</table>

**Description**

The `rng` keyword is used to indicate a choice of random number generator.

**Default Behavior**

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

**Usage Tips**

The default is recommended, as the Mersenne twister is a higher quality random number generator.

**Examples**

```
method sampling
  sample_type lhs
  samples = 10
  seed = 98765
  rng rnum2
```

```
mt19937
```

- **Keywords Area**
- **method**
- **global_reliability**
- **rng**
- **mt19937**

Generates random numbers using the Mersenne twister
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on Wikipedia: \url{http://en.wikipedia.org/wiki/Mersenne_twister}.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```plaintext
method sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

rnum2

- Keywords Area
- method
- global_reliability
- rng
- rnum2

Generates pseudo-random numbers using the Pecos package.

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`. 
6.2. METHOD

Examples

```plaintext
method sampling
  sample_type lhs
  samples = 10
  seed = 98765
  rng rnum2
```

`response_levels`

- **Keywords Area**
- **method**
- **global_reliability**
- **response_levels**

Values at which to estimate desired statistics for each response

**Specification**

*Alias*: none

*Argument(s)*: REALLIST

*Default*: No CDF/CCDF probabilities/reliabilities to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_response_levels</td>
<td>compute</td>
<td>Number of values at which to estimate desired statistics for each response</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>Selection of statistics to compute at each response level</td>
</tr>
</tbody>
</table>

**Description**

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

**Default Behavior**

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

**Expected Outputs**

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or...
The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips
The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples
For example, specifying a `response_level` of 52.3 followed with `compute_probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```plaintext
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```plaintext
method,
  sampling,
    samples = 100 seed = 1
  complementary_distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
    6.0e+04 6.5e+04 7.0e+04
    3.5e+05 4.0e+05 4.5e+05
variables,
  normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas = 12., 30.
    betas = 250., 590.
    descriptors = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
    num_pairs = 3 4
    abscissas = 5 8 10 .1 .2 .3 .4
    counts = 17 21 0 12 24 12 0
    descriptors = 'TF1h' 'TF2h'
  histogram_point_uncertain
    real = 1
    num_pairs = 2
    abscissas = 3 4
    counts = 1 1
    descriptors = 'TF3h'
interface,
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'
```
6.2. METHOD

responses,
response_functions = 3
no_gradients
no_hessians

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.6000000000e+11</td>
<td>5.3601733194e-12</td>
</tr>
<tr>
<td>3.6000000000e+11</td>
<td>4.0000000000e+11</td>
<td>4.2500000000e-12</td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>4.4000000000e+11</td>
<td>3.7500000000e-12</td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>5.419614379e+11</td>
<td>2.2557612778e-12</td>
</tr>
</tbody>
</table>

PDF for response_fn_2:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.643154744e+04</td>
<td>6.0000000000e+04</td>
<td>2.8742313192e-05</td>
</tr>
<tr>
<td>6.0000000000e+04</td>
<td>6.5000000000e+04</td>
<td>6.4000000000e-05</td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>7.0000000000e+04</td>
<td>4.0000000000e-05</td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>7.8702465755e+04</td>
<td>1.0341896485e-05</td>
</tr>
</tbody>
</table>

PDF for response_fn_3:

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3796737090e+05</td>
<td>3.5000000000e+05</td>
<td>4.2844660868e-06</td>
</tr>
<tr>
<td>3.5000000000e+05</td>
<td>4.0000000000e+05</td>
<td>8.6000000000e-06</td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>4.5000000000e+05</td>
<td>1.8000000000e-06</td>
</tr>
</tbody>
</table>

Level mappings for each response function:
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6000000000e+11</td>
<td>5.5000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+11</td>
<td>3.8000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4000000000e+11</td>
<td>2.3000000000e-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0000000000e+04</td>
<td>6.1000000000e-01</td>
<td></td>
</tr>
<tr>
<td>6.5000000000e+04</td>
<td>2.9000000000e-01</td>
<td></td>
</tr>
<tr>
<td>7.0000000000e+04</td>
<td>9.0000000000e-02</td>
<td></td>
</tr>
</tbody>
</table>

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5000000000e+05</td>
<td>5.2000000000e-01</td>
<td></td>
</tr>
<tr>
<td>4.0000000000e+05</td>
<td>9.0000000000e-02</td>
<td></td>
</tr>
<tr>
<td>4.5000000000e+05</td>
<td>0.0000000000e+00</td>
<td></td>
</tr>
</tbody>
</table>

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.
num_response_levels
  • Keywords Area
  • method
  • global_reliability
  • response_levels
  • num_response_levels

Number of values at which to estimate desired statistics for each response

Specification
Alias: none
  Argument(s): INTEGERLIST
  Default: response_levels evenly distributed among response functions

Description
The num_response_levels keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

  Default Behavior
  If num_response_levels is not specified, the response_levels will be evenly distributed among the responses.

  Expected Outputs
  The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5

compute
  • Keywords Area
  • method
  • global_reliability
  • response_levels
  • compute

Selection of statistics to compute at each response level
### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** probabilities

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>probabilities</td>
<td>Computes probabilities associated with response levels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional</th>
<th>gen_reliabilities</th>
<th>Computes generalized reliabilities associated with response levels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>system</td>
<td>Compute system reliability (series or parallel)</td>
</tr>
</tbody>
</table>

### Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

**Default Behavior**

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

**Expected Output**

The type of statistics specified by `compute` will be reported for each response level.

**Usage Tips**

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach. CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

### Examples

```bash
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11 6.0e+04 6.5e+04 7.0e+04 3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
    probabilities
```
CHAPTER 6. KEYWORDS AREA

- Keywords Area
- method
- global_reliability
- response_levels
- compute
- probabilities

Computes probabilities associated with response levels

**Specification**

Alias: none

Argument(s): none

**Description**

The **probabilities** keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If **response_levels** is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the **probabilities** keyword should be specified in conjunction with the **compute** keyword.

**Expected Outputs**

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

**Examples**

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                6.0e+04 6.5e+04 7.0e+04
                3.5e+05 4.0e+05 4.5e+05
    compute probabilities

gen_reliabilities
```

- Keywords Area
- method
- global_reliability
- response_levels
- compute
6.2. METHOD

- **gen_reliabilities**

  Computes generalized reliabilities associated with response levels

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

**Default Behavior**

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

**Expected Outputs**

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

**Examples**

```plaintext
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities

system

- Keywords Area
- method
- global_reliability
- response_levels
- compute
- system

  Compute system reliability (series or parallel)

**Specification**

**Alias:** none  
**Argument(s):** none
### Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response. For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

- **series**
  - Keywords Area
  - method
  - global_reliability
  - response_levels
  - compute
  - system
  - series

  Aggregate response statistics assuming a series system

### Specification

**Alias:** none

**Argument(s):** none

### Description

See parent keyword `system` for description.
6.2. METHOD

parallel

• Keywords Area
• method
• global_reliability
• response_levels
• compute
• system
• parallel

Aggregate response statistics assuming a parallel system

Specification
Alias: none
Argument(s): none

Description
See parent keyword system for description.

max_iterations

• Keywords Area
• method
• global_reliability
• max_iterations

Stopping criterion based on number of iterations

Topics
This keyword is related to the topics:

• method_independent_controls

Specification
Alias: none
Argument(s): INTEGER
Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.
convergence_tolerance

- Keywords Area
- method
- global_reliability
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL
Default: 1.e-4

Description

The convergence_tolerance specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by convergence_tolerance, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations

- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of convergence_tolerance approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., convergence_tolerance = 1.e-6 will result in approximately six digits of accuracy in the final objective function).

- NL2SOL: See nl2sol
distribution

- Keywords Area
- method
- global_reliability
- distribution

Selection of cumulative or complementary cumulative functions

Specification

Alias: none
Argument(s): none
Default: cumulative (CDF)

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td></td>
<td>cumulative</td>
<td>Computes statistics according to cumulative functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>complementary</td>
<td>Computes statistics according to complementary cumulative functions</td>
</tr>
</tbody>
</table>

Description

The distribution keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the distribution keyword is present, it must be accompanied by either cumulative or complementary. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```plaintext
method
  sampling
    sample_type lhs
    samples = 10
  distribution cumulative
```
CHAPTER 6. KEYWORDS AREA

**cumulative**

- Keywords Area
- method
- global_reliability
- distribution
- cumulative

Computes statistics according to cumulative functions

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

**Default Behavior**

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

**Expected Outputs**

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

**Examples**

```plaintext
method
  sampling
    sample_type lhs
    samples - 10
  distribution cumulative
```

**complementary**

- Keywords Area
- method
- global_reliability
- distribution
- complementary

Computes statistics according to complementary cumulative functions
6.2. METHOD

Specification

Alias: none
  Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the complementary keyword must appear in conjunction with the distribution keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```dakota
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

**probability_levels**

- Keywords Area
- method
- global_reliability
- probability_levels

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none
  Argument(s): REALLIST
  Default: No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>num_probability_levels</td>
<td>Specify which probability_levels correspond to which response</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If probability_levels are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the probability_levels and/or response_levels in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```dakota
method,  
sampling,  
samples = 100 seed = 1  
complementary distribution  
probability_levels = 1. .66 .33 0.  
  1.8 .5 0.  
  1.3 .2 0.  
variables,  
  normal_uncertain = 2  
  means = 248.89, 593.33  
  std_deviations = 12.4, 29.7  
  descriptors = 'TF1n' 'TF2n'  
  uniform_uncertain = 2  
  lower_bounds = 199.3, 474.63  
  upper_bounds = 298.5, 712.  
  descriptors = 'TF1u' 'TF2u'  
  weibull_uncertain = 2  
  alphas = 12., 30.  
  betas = 250., 590.  
  descriptors = 'TF1w' 'TF2w'  
  histogram_bin_uncertain = 2  
  num_pairs = 3 4  
  abscissas = 5 8 10 .1 .2 .3 .4  
  counts = 17 21 0 12 24 12 0  
  descriptors = 'TF1h' 'TF2h'  
  histogram_point_uncertain  
  real = 1  
  num_pairs = 2  
  abscissas = 3 4  
  counts = 1 1  
  descriptors = 'TF3h'  
interface,  
  system asynch evaluation_concurrency = 5  
  analysis_driver = 'text_book'  
responses,  
  response_functions = 3  
  no_gradients  
  no_hessians
```
Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

### Probability Density Function (PDF) histograms for each response function:

**PDF for response_fn_1:**

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>3.4221494996e+11</td>
<td>5.1384774972e-12</td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>4.0634975300e+11</td>
<td>5.145123115e-12</td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>5.4196113479e+11</td>
<td>2.4334239039e-12</td>
</tr>
</tbody>
</table>

**PDF for response_fn_2:**

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.643154744e+04</td>
<td>5.6511827775e+04</td>
<td>1.9839945149e-05</td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>6.1603813790e+04</td>
<td>5.8916108390e-05</td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>7.8702465755e+04</td>
<td>2.9242071306e-05</td>
</tr>
</tbody>
</table>

**PDF for response_fn_3:**

<table>
<thead>
<tr>
<th>Bin Lower</th>
<th>Bin Upper</th>
<th>Density Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.379673709e+05</td>
<td>3.6997214153e+05</td>
<td>5.3028385253e-06</td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.8100966235e+05</td>
<td>9.0600055634e-06</td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>4.4111498127e+05</td>
<td>3.3274925348e-06</td>
</tr>
</tbody>
</table>

### Level mappings for each response function:

**Complementary Cumulative Distribution Function (CCDF) for response_fn_1:**

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7604749078e+11</td>
<td>1.00000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4221494996e+11</td>
<td>6.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0634975300e+11</td>
<td>3.30000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4196113479e+11</td>
<td>0.00000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complementary Cumulative Distribution Function (CCDF) for response_fn_2:**

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.643154744e+04</td>
<td>1.00000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6511827775e+04</td>
<td>8.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1603813790e+04</td>
<td>5.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.8702465755e+04</td>
<td>0.00000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complementary Cumulative Distribution Function (CCDF) for response_fn_3:**

<table>
<thead>
<tr>
<th>Response Level</th>
<th>Probability Level</th>
<th>Reliability Index</th>
<th>General Rel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.379673709e+05</td>
<td>1.00000000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6997214153e+05</td>
<td>3.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8100966235e+05</td>
<td>2.00000000000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4111498127e+05</td>
<td>0.00000000000e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

**num_probability_levels**
Keywords Area

- method
- global_reliability
- probability_levels
- num_probability_levels

Specify which probability_levels correspond to which response

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** probability_levels evenly distributed among response functions

**Description**

See parent page

gen_reliability_levels

- Keywords Area
- method
- global_reliability
- gen_reliability_levels

Specify generalized reliability levels at which to estimate the corresponding response value

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Default:** No CDF/CCDF response levels to compute

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>num_gen_reliability_levels</td>
<td>Specify which gen_reliability_levels correspond to which response</td>
</tr>
</tbody>
</table>

**Description**

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).
6.2. METHOD

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

\texttt{num\_gen\_reliability\_levels}

- Keywords Area
- method
- global\_reliability
- gen\_reliability\_levels
- num\_gen\_reliability\_levels

Specify which gen\_reliability\_levels correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen\_reliability\_levels evenly distributed among response functions

Description

See parent page

\texttt{model\_pointer}

- Keywords Area
- method
- global\_reliability
- model\_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block\_pointer
CHAPTER 6. KEYWORDS AREA

Specification

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The `model_pointer` is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
          0.1 0.2 0.6
          0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'
```
6.2. METHOD

interface
    id_interface = 'II'
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
    response_functions = 3
    no_gradients
    no_hessians

6.2.65 fsu_quasi_mc

    • Keywords Area

    • method

    • fsu_quasi_mc

    Design of Computer Experiments - Quasi-Monte Carlo sampling

Topics

This keyword is related to the topics:

    • package_fсудace

    • design_and_analysis_of_computer_experiments

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group sequence type (Group 1)</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required(Choose One)</td>
<td>halton</td>
<td>Generate samples from a Halton sequence</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>hammersley</td>
<td>Use Hammersley sequences</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>latinize</td>
<td>Adjust samples to improve the discrepancy of the marginal distributions</td>
<td></td>
</tr>
</tbody>
</table>
## Keywords Area

### Description

Quasi-Monte Carlo methods produce low discrepancy sequences, especially if one is interested in the uniformity of projections of the point sets onto lower dimensional faces of the hypercube (usually 1-D: how well do the marginal distributions approximate a uniform?)

This method generates sets of uniform random variables on the interval [0,1]. If the user specifies lower and upper bounds for a variable, the [0,1] samples are mapped to the [lower, upper] interval.

The user must first choose the sequence type:

| Optional | quality_metrics | Calculate metrics to assess the quality of quasi-Monte Carlo samples |
| Optional | variance_based_decomp | Activates global sensitivity analysis based on decomposition of response variance into contributions from variables |
| Optional | samples | Number of samples for sampling-based methods |
| Optional | fixed_sequence | Reuse the same sequence and samples for multiple sampling sets |
| Optional | sequence_start | Choose where to start sampling the sequence |
| Optional | sequence_leap | Specify how often the sequence is sampled |
| Optional | prime_base | The prime numbers used to generate the sequence |
| Optional | max_iterations | Stopping criterion based on number of iterations |
| Optional | model_pointer | Identifier for model block to be used by a method |
6.2. METHOD

- halton or
- hammersley

Then three keywords are used to define the sequence and how it is sampled:

- prime_base
- sequence_start
- sequence_leap

Each of these has defaults, so specification is optional.

Theory

The quasi-Monte Carlo sequences of Halton and Hammersley are deterministic sequences determined by a set of prime bases. Generally, we recommend that the user leave the default setting for the bases, which are the lowest primes. Thus, if one wants to generate a sample set for 3 random variables, the default bases used are 2, 3, and 5 in the Halton sequence. To give an example of how these sequences look, the Halton sequence in base 2 starts with points 0.5, 0.25, 0.75, 0.125, 0.625, etc. The first few points in a Halton base 3 sequence are 0.33333, 0.66667, 0.11111, 0.44444, 0.77777, etc. Notice that the Halton sequence tends to alternate back and forth, generating a point closer to zero then a point closer to one. An individual sequence is based on a radix inverse function defined on a prime base. The prime base determines how quickly the [0,1] interval is filled in. Generally, the lowest primes are recommended.

The Hammersley sequence is the same as the Halton sequence, except the values for the first random variable are equal to 1/N, where N is the number of samples. Thus, if one wants to generate a sample set of 100 samples for 3 random variables, the first random variable has values 1/100, 2/100, 3/100, etc. and the second and third variables are generated according to a Halton sequence with bases 2 and 3, respectively.

For more information about these sequences, see [43], [44], and [1].

See Also

These keywords may also be of interest:

- dace
- fsu_cvt
- psuade_moat

halton

- Keywords Area
- method
- fsu_quasi_nc
- halton

Generate samples from a Halton sequence
CHAPTER 6. KEYWORDS AREA

Topics
This keyword is related to the topics:

- package fsudace

Specification
Alias: none
Argument(s): none

Description
The quasi-Monte Carlo sequences of Halton are deterministic sequences determined by a set of prime bases. These sequences generate random numbers with the goal of filling a unit hypercube uniformly.

Generally, we recommend that the user leave the default setting for the bases, which are the lowest primes. Thus, if one wants to generate a sample set for 3 random variables, the default bases used are 2, 3, and 5 in the Halton sequence. To give an example of how these sequences look, the Halton sequence in base 2 starts with points 0.5, 0.25, 0.75, 0.125, 0.625, etc. The first few points in a Halton base 3 sequence are 0.33333, 0.66667, 0.11111, 0.44444, 0.77777, etc. Notice that the Halton sequence tends to alternate back and forth, generating a point closer to zero then a point closer to one. An individual sequence is based on a radix inverse function defined on a prime base. The prime base determines how quickly the [0,1] interval is filled in.

Theory
For more information about these sequences, see [43], [44], and [1].

hammersley

- Keywords Area
- method
- fsu_quasi_mc
- hammersley

Use Hammersley sequences

Topics
This keyword is related to the topics:

- package fsudace
- design_and_analysis_of_computer_experiments

Specification
Alias: none
Argument(s): none
6.2. METHOD

Description
The Hammersley sequence is the same as the Halton sequence, except the values for the first random variable are equal to 1/N, where N is the number of samples. Thus, if one wants to generate a sample set of 100 samples for 3 random variables, the first random variable has values 1/100, 2/100, 3/100, etc. and the second and third variables are generated according to a Halton sequence with bases 2 and 3, respectively.

See Also
These keywords may also be of interest:
- fsu_quasi_mc

latinize
- Keywords Area
- method
- fsu_quasi_mc
- latinize

Adjust samples to improve the discrepancy of the marginal distributions

Specification
Alias: none
Argument(s): none
Default: No latinization

Description
The latinize control takes the samples and "latinizes" them, meaning that each original sample is moved so that it falls into one strata or bin in each dimension as in Latin Hypercube sampling. The default setting is NOT to latinize. However, one may be interested in doing this in situations where one wants better discrepancy of the 1-dimensional projections (the marginal distributions).

quality_metrics
- Keywords Area
- method
- fsu_quasi_mc
- quality_metrics

Calculate metrics to assess the quality of quasi-Monte Carlo samples

Topics
This keyword is related to the topics:
- package_fsudace
**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** No quality_metrics

**Description**

`quality_metrics` calculates four quality metrics relating to the volumetric spacing of the samples. The four quality metrics measure different aspects relating to the uniformity of point samples in hypercubes. Desirable properties of such point samples are:

- are the points equally spaced  
- do the points cover the region  
- and are they isotropically distributed  
- with no directional bias in the spacing

The four quality metrics we report are:

- \( h \): the point distribution norm, which is a measure of uniformity of the point distribution  
- \( \chi \): a regularity measure, and provides a measure of local uniformity of a set of points  
- \( \tau \): the second moment trace measure  
- \( d \): the second moment determinant measure

All of these values are scaled so that smaller is better (the smaller the metric, the better the uniformity of the point distribution).

**Examples**

Complete explanation of these measures can be found in [38].

`variance_based_decomp`

- Keywords Area  
- method  
- fsu_quasi_mc  
- variance_based_decomp

Activates global sensitivity analysis based on decomposition of response variance into contributions from variables

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** no variance-based decomposition
6.2. **METHOD**

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>drop_tolerance</td>
<td>Suppresses output of sensitivity indices with values lower than this tolerance</td>
</tr>
</tbody>
</table>

**Description**

Dakota can calculate sensitivity indices through variance based decomposition using the keyword `variance_based_decomp`. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

**Default Behavior**

Because of the computational cost, `variance_based_decomp` is turned off as a default.

If the user specified a number of samples, N, and a number of nondeterministic variables, M, variance-based decomposition requires the evaluation of N*(M+2) samples. **Note that specifying this keyword will increase the number of function evaluations above the number requested with the `samples` keyword since replicated sets of sample values are evaluated.**

**Expected Outputs**

When `variance_based_decomp` is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response.

**Usage Tips**

To obtain sensitivity indices that are reasonably accurate, we recommend that N, the number of samples, be at least one hundred and preferably several hundred or thousands.

**Examples**

```
method,
sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
```

**Theory**

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [73]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, $S_i$, means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [73] and [87].

**drop_tolerance**

- **Keywords Area**
- **method**
• `fsu_quasi_mc`  
• `variance_based_decomp`  
• `drop_tolerance`

Suppresses output of sensitivity indices with values lower than this tolerance

**Specification**

**Alias:** none  
**Argument(s):** REAL  
**Default:** All VBD indices displayed

**Description**

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

**Default Behavior**

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

**Usage Tips**

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be diagonal_covariance or full_covariance, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

**Examples**

```plaintext
method,  
  sampling  
    sample_type lhs  
    samples = 100  
    variance_based_decomp  
    drop_tolerance = 0.001
```

**samples**

• **Keywords Area**  
• **method**  
• `fsu_quasi_mc`  
• **samples**

Number of samples for sampling-based methods

**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** 0 (or min req for surrogate build)
6.2. **METHOD**

**Description**

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

**Default Behavior**

By default, Dakota will use the minimum number of samples required by the chosen method.

**Usage Tips**

To obtain linear sensitivities or to construct a linear response surface, at least \( \text{dim}+1 \) samples should be used, where "\( \text{dim} \)" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least \( (\text{dim}+1)(\text{dim}+2)/2 \) samples are needed. For uncertainty quantification, we recommend at least \( 10^*\text{dim} \) samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be \( N^*(\text{dim}+2) \).

**Examples**

```yaml
method
  sampling
    sample_type lhs
    samples = 20
```

**fixed_sequence**

- **Keywords Area**
- **method**
- **fsu_quasi_mc**
- **fixed_sequence**

Reuse the same sequence and samples for multiple sampling sets

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** sequence not fixed: sampling patterns are variable among multiple QMC runs

**Description**

The `fixed_sequence` control is similar to `fixed_seed` for other sampling methods. If `fixed_sequence` is specified, the user will get the same sequence (meaning the same set of samples) for subsequent calls of the QMC sampling method (for example, this might be used in a surrogate based optimization method or a parameter study where one wants to fix the uncertain variables).

**sequence_start**

- **Keywords Area**
- **method**
- **fsu_quasi_mc**
- **sequence_start**

Choose where to start sampling the sequence
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): INTEGERLIST

Default: Vector of zeroes

Description

sequence_start determines where in the sequence the samples will start.

The default sequence_start is a vector with 0 for each variable, specifying that each sequence start with the first term.

Examples

For example, for the Halton sequence in base 2, if the user specifies sequence_start = 2, the sequence would not include 0.5 and 0.25, but instead would start at 0.75.

sequence_leap

- Keywords Area
- method
- fsu_quasi_mc
- sequence_leap

Specify how often the sequence is sampled

Specification

Alias: none

Argument(s): INTEGERLIST

Default: Vector of ones

Description

sequence_leap controls the "leaping" of terms in the sequence. The default is 1 for each variable, meaning that each term in the sequence be returned.

Examples

If the user specifies a sequence_leap of 2 for a variable, the points returned would be every other term from the QMC sequence.

Theory

The advantage to using a leap value greater than one is mainly for high-dimensional sets of random deviates. In this case, setting a leap value to the next prime number larger than the largest prime base can help maintain uniformity when generating sample sets for high dimensions. For more information about the efficacy of leaped Halton sequences, see[71].
6.2. METHOD

prime_base

- Keywords Area
- method
- fsu_quasi_mc
- prime_base

The prime numbers used to generate the sequence

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Default:** Vector of the first s primes for s-dimensions in Halton, First (s-1) primes for Hammersley

**Description**

It is recommended that the user not specify this and use the default values.

- For the Halton sequence, the default bases are primes in increasing order, starting with 2, 3, 5, etc.
- For the Hammersley sequence, the user specifies (s-1) primes if one is generating an s-dimensional set of random variables.

**max_iterations**

- Keywords Area
- method
- fsu_quasi_mc
- max_iterations

Stopping criterion based on number of iterations

**Topics**

This keyword is related to the topics:

- method_independent_controls

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 100 (exceptions: fsu_cvt, local_reliability: 25; global_{reliability, interval_est, evidence} / efficient_global: 25*n)

**Description**

The maximum number of iterations (default: 100). See also max_function_evaluations.
model_pointer

- Keywords Area
- method
- fsu_quasi_mc
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:
- block_pointer

Specification
Alias: none
- Argument(s): STRING
- Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.
See block_pointer for details about pointers.

Examples
environment
   tabular_graphics_data
   method_pointer = 'UQ'

method
   id_method = 'UQ'
   model_pointer = 'SURR'
   sampling,
     samples = 10
     seed = 98765 rng rnum2
     response_levels = 0.1 0.2 0.6
     0.1 0.2 0.6
     0.1 0.2 0.6
   sample_type lhs
   distribution cumulative

model
   id_model = 'SURR'
6.2. METHOD

surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.66 vector_parameter_study

- **Keywords Area**
- **method**
- **vector_parameter_study**

  Samples variables along a user-defined vector

**Topics**
This keyword is related to the topics:
- **parameter_studies**

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
</table>
CHAPTER 6. KEYWORDS AREA

### Description

Dakota’s vector parameter study computes response data sets at selected intervals along a vector in parameter space. It is often used for single-coordinate parameter studies (to study the effect of a single variable on a response set), but it can be used more generally for multiple coordinate vector studies (to investigate the response variations along some n-dimensional vector such as an optimizer search direction).

**Default Behavior**

By default, the multidimensional parameter study operates over all types of variables.

**Expected Outputs**

A multidimensional parameter study produces a set of responses for each parameter set that is generated.

**Usage Tips**

**Group 1** is used to define the vector along which the parameters are varied. Both cases also rely on the variables specification of an initial value, through:

- the `initial_point` keyword
- the `initial_state` keyword
- relying on the default initial value, based on the rest of the variables specification

From the initial value, the vector can be defined using one of the two keyword choices. Once the vector is defined, the samples are then fully specified by `num_steps`.

### Examples

The following example is a good comparison to the examples on `multidim.parameter_study` and `centered.parameter_study`.

```bash
# tested on Dakota 6.0 on 140501
environment
tabular_data
tabular_data_file = 'rosen_vector.dat'

method
vector-parameter_study
num_steps = 10
```
6.2. METHOD

```
final_point =  2.0  2.0
model
  single

variables
  continuous_design = 2
  initial_point = -2.0 -2.0
  descriptors = 'x1'  'x2'

interface
  analysis_driver = 'rosenbrock'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians

See Also

These keywords may also be of interest:

- centered_parameter_study
- multidim_parameter_study
- list_parameter_study

final_point

- Keywords Area
- method
- vector_parameter_study
- final_point

Final variable values defining vector in vector parameter study

Specification

Alias: none
Argument(s): REALLIST

Description

The final_point keyword is used to define the final values for each variable on the vector to be used in the vector parameter study. The vector's direction and magnitude are determined by the initial value from the variables specification, and the final_point.

Default Behavior

The user is required to specify either final_point or step_vector. There is no default definition for the vector.

Usage Tips

The actual points are determined based on this vector and the number of points chosen is given in num_points.
Examples

```
method
   vector_parameter_study
      num_steps = 10
      final_point = 2.0  2.0

step_vector
   • Keywords Area
   • method
   • vector_parameter_study
   • step_vector

Size of step for each variable
```

Specification

Alias: none

Argument(s): REALLIST

Description

The `step_vector` keyword specifies how much each variable will be incremented in a single step. `step_vector` works in conjunction with `num_steps`, which determines the number of steps taken during the `vector_parameter_study`. If instead of `step_vector`, `final_point` is specified with `num_steps`, Dakota will infer the step sizes.

Entries in the `step_vector` are the actual amounts by which continuous and range variables are incremented. For set variables, `step_vector` entries are interpreted as indexes into the underlying set.

Default Behavior

The user is required to specify either `final_point` or `step_vector`. There is no default definition for the vector.

Examples

```
variables
   continuous_design 1
      initial_point 1.0
      descriptors 'xl'
   discrete_design_set
      string 1
         elements 'bar' 'baz' 'foo' 'fuzz'
         initial_point 'bar'
         descriptors 's1'

method
   vector_parameter_study
      num_steps = 3
      # Add 2.0 to xl and increment s1 by 1 element in each step
      step_vector = 2.0  1
```
6.2. METHOD

num_steps

- Keywords Area
- method
- vector_parameter_study
- num_steps

Number of sampling steps along the vector in a vector parameter study

Specification

Alias: none
Argument(s): INTEGER

Description

num_steps defines the number of steps that are taken in the direction of the vector. The magnitude of each step is determined in conjunction with the rest of the method specification.

Default Behavior

The user is required to specify num_steps for a vector parameter study. There is no default value.

This study performs function evaluations at both ends, making the total number of evaluations equal to num_steps+1.

Usage Tips

The study has stringent requirements on performing appropriate steps with any discrete range and discrete set variables. A num_steps specification must result in discrete range and set index steps that are integers: no remainder is currently permitted in the integer step calculation and no rounding to integer steps will occur.

Examples

```plaintext
method
  vector_parameter_study
  num_steps = 10
  final_point = 2.0 2.0
```

model_pointer

- Keywords Area
- method
- vector_parameter_study
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.

See block_pointer for details about pointers.

Examples

environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
              0.1 0.2 0.6
              0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'
6.2. METHOD

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.67 list_parameter_study

- Keywords Area
- method
- list_parameter_study

Samples variables as a specified values

Topics

This keyword is related to the topics:
- parameter_studies

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required(Choose One)</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_of_points</td>
<td></td>
<td>list_of_points</td>
<td>List of variable values to evaluate in a list parameter study</td>
</tr>
<tr>
<td>import_points_file</td>
<td></td>
<td>import_points_file</td>
<td>File containing list of variable values to evaluate in a list parameter study</td>
</tr>
<tr>
<td>model_pointer</td>
<td></td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

Description

Dakota's list parameter study allows for evaluations at user selected points of interest.

Default Behavior

By default, the list parameter study operates over all types of variables.
The number of real values in the list_of_points specification or file referenced by import_points_file must be a multiple of the total number of variables (including continuous and discrete types) contained in the variables specification.

**Expected Outputs**

A list parameter study produces a set of responses for each parameter set that is specified.

**Usage Tips**

- This parameter study simply performs simulations for the first parameter set (the first $n$ entries in the list), followed by the next parameter set (the next $n$ entries), and so on, until the list of points has been exhausted.
- Since the initial values from the variables specification will not be used, they need not be specified.
- When the points are specified in the Dakota input file using the list_of_points keyword, discrete set values must be referred to using 0-based indexes, and not the values themselves. However, when using import_points_file, refer to discrete set values directly, not by index.
- For both list_of_points and import_points_file, Dakota expects the values for each evaluation to be ordered by type. The type ordering matches that of the variables section of this Reference Manual and of the listing in the Parameters file format section of the Dakota User’s Manual[4]. When multiple variables are present for a single type, the ordering within that type must match the order specified by the user in the variables section of her input file.

**Examples**

This shows the method and variables block of a Dakota input file that runs a list_parameter_study.

```dakota
method
  list_parameter_study
  list_of_points =
  3.1e6 0.0029 0.31
  3.2e6 0.0028 0.32
  3.3e6 0.0027 0.34
  3.3e6 0.0026 0.36

variables
  continuous_design = 3
  descriptors = 'E' 'MASS' 'DENSITY'
```

Note that because of the way Dakota treats whitespace, the above example is equivalent to:

```dakota
method
  list_parameter_study
  list_of_points =
  3.1e6 0.0029 0.31
  3.2e6 0.0028
  3.3e6
  0.0027
  0.32
  3.3e6
  0.0026
  0.36

variables
  continuous_design = 3
  descriptors = 'E' 'MASS' 'DENSITY'
```

Although the first example is much more readable.

And here’s a full input file:

```dakota
# tested on Dakota 6.0 on 140501
environment
  tabular_data
  tabular_data_file 'List_param_study.dat'
```
6.2. METHOD

method
list_parameter_study
list_of_points = 0.1 0.1
0.2 0.1
0.3 0.0
0.3 1.0

model
single

variables
active design
continuous_design = 2
  descriptors 'x1' 'x2'
continuous_state = 1
  descriptors = 'constant1'
  initial_state = 100

interface
analysis_drivers 'text_book'
  fork
  asynchronous
    evaluation_concurrency 2

responses
response_functions = 1
no_gradients
no_hessians

This example illustrates the list_parameter_study.

- The function evaluations are independent, so any level of evaluation_concurrency can be used
- Default behavior for parameter studies is to iterate on all variables. However, because active design is specified, this study will only iterate on the continuous_design variables.

See Also

These keywords may also be of interest:

- centered_parameter_study
- multidim_parameter_study
- vector_parameter_study

list_of_points

- Keywords Area
- method
- list_parameter_study
- list_of_points

List of variable values to evaluate in a list parameter study
Specification

Alias: none
Argument(s): REALLIST

Description

The list_of_points keyword allows the user to specify, in a freeform format, a list of variable values at which to compute a model response.

Default Behavior

The user is required to provide a list of points for a list parameter study either by specifying it with list_of_points or by providing a file from which such a list can be read via import_points_file. There is no default list of points.

Usage Tips

The number of values in the list must be an integer multiple of the number of variables. Dakota will verify that this condition is met.

Examples

```plaintext
method
  list_parameter_study
    list_of_points =
      3.1e6  0.0029  0.31
      3.2e6  0.0028  0.32
      3.3e6  0.0027  0.34
      3.3e6  0.0026  0.36
```

import_points_file

- Keywords Area
- method
- list_parameter_study
- import_points_file

File containing list of variable values to evaluate in a list parameter study

Specification

Alias: none
Argument(s): STRING
Default: no point import from a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular_format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
</tbody>
</table>
6.2. METHOD

<table>
<thead>
<tr>
<th>Optional</th>
<th>custom.annotated</th>
<th>Selects custom-annotated tabular file format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td></td>
<td>active.only</td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

### Description

The `import_points_file` specifies a file containing a list of variable values at which to compute a model response.

**Default Behavior**

By default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the `import_points_file` must be in one of three formats:

- annotated (default)
- custom.annotated
- freeform

### Examples

```bash
top_level
  method
    list_parameter_study
      import_points_file = 'dakota_pstudy.3.dat'
```

**annotated**

- **Keywords Area**
- **method**
- **list_parameter_study**
- **import_points_file**
- **annotated**

Selects annotated tabular file format

### Topics

This keyword is related to the topics:

- **file_formats**
**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated

**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

**Usage Tips**

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify `custom_annotated header eval_id`

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

**Examples**

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.00019996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`custom_annotated`

- **Keywords Area**
- **method**
- **list_parameter_study**
- **import_points_file**
- **custom_annotated**

Selects custom-annotated tabular file format
6.2. METHOD

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nlneq_con_1</th>
<th>nlneq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

header

- Keywords Area
- method
- list_parameter_study
- import_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

See description of parent custom_annotated

**eval_id**

- Keywords Area
- method
- list_parameter_study
- import_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none
**Argument(s):** none
6.2. METHOD

**Description**

See description of parent `custom_annotated`

`interface_id`
- Keywords Area
- method
- `list_parameter_study`
- `import_points_file`
- `custom_annotated`
- `interface_id`

Enable interface ID column in custom-annotated tabular file

**Specification**

Alias: none
Argument(s): none

**Description**

See description of parent `custom_annotated`

`freeform`
- Keywords Area
- method
- `list_parameter_study`
- `import_points_file`
- `freeform`

Selects freeform file format

**Topics**

This keyword is related to the topics:
- `file_formats`

**Specification**

Alias: none
Argument(s): none
**Description**

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

**Default Behavior**

The *annotated* format is the default for tabular export/import. To change this behavior, specify *freeform* in the relevant export/import context.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to *annotated* format, though *freeform* remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In *freeform*, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the *freeform* option.

**Examples**

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9   1.1   0.0002  0.26   0.76
0.90009 1.1 0.0001996404857 0.2601620081 0.759955
0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

**active_only**

- *Keywords Area*

- *method*

- *list_parameter_study*

- *import_points_file*

- *active_only*

Import only active variables from tabular data file

**Topics**

This keyword is related to the topics:

- *file_formats*
6.2. METHOD

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

**model_pointer**

- **Keywords Area**
- **method**
- **list_parameter_study**
- **model_pointer**

Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The **model_pointer** is used to specify which **model** block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a **model** block in the Dakota input file that has a corresponding **id_model** with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a **model-pointer** for each method is imperative.

See **block_pointer** for details about pointers.
Examples

environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
0.1 0.2 0.6
0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system asynch evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

6.2.68 centered_parameter_study

- Keywords Area
- method
- centered_parameter_study

Samples variables along points moving out from a center point
6.2. METHOD

Topics
This keyword is related to the topics:

- parameter_studies

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>step_vector</td>
<td>Size of steps to be taken in each dimension of a centered parameter study</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>steps_per_variable</td>
<td>Number of steps to take in each dimension of a centered parameter study</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>model_pointer</td>
<td>Identifier for model block to be used by a method</td>
</tr>
</tbody>
</table>

Description

Dakota’s centered parameter study computes response data sets along multiple coordinate-based vectors, one per parameter, centered about the initial values from the variables specification. This is useful for investigation of function contours with respect to each parameter individually in the vicinity of a specific point (e.g., post-optimality analysis for verification of a minimum), thereby avoiding the cost associated with a multidimensional grid.

Default Behavior

By default, the centered parameter study operates over all types of variables.

The centered_parameter_study takes steps along each orthogonal dimension. Each dimension is treated independently. The number of steps are taken in each direction, so that the total number of points in the parameter study is $1 + 2 \sum n$.

Expected Outputs

A centered parameter study produces a set of responses for each parameter set that is generated.

Examples

The following example is a good comparison to the examples on multidim_parameter_study and vector_parameter_study.

# tested on Dakota 6.0 on 140501 environment
tabular_data
    tabular_data_file = 'rosen_centered.dat'

method
    centered_parameter_study
    steps_per_variable = 5 4
    step_vector = 0.4 0.5

model
    single

variables
    continuous_design = 2
    initial_point = 0 0
    descriptors = 'x1' 'x2'

interface
    analysis_driver = 'rosenbrock'
    fork

responses
    response_functions = 1
    no_gradients
    no_hessians

See Also

These keywords may also be of interest:

* multidim_parameter_study
* list_parameter_study
* vector_parameter_study

step_vector

* Keywords Area
* method
* centered_parameter_study
* step_vector

Size of steps to be taken in each dimension of a centered parameter study

Specification

Alias: none

Argument(s): REALLIST

Description

The step_vector keyword defines the individual step size in each dimension, treated separately.
The user is required to define the number of step sizes for a centered parameter study. There are no default values.

Steps are taken in the plus and minus directions, and are defined in either actual values (continuous and discrete range) or index offsets (discrete set).

**Examples**

```plaintext
method
  centered_parameter_study
  steps_per_variable = 5 4
  step_vector = 0.4 0.5
```

**steps_per_variable**

- **Keywords Area**
- **method**
- **centered_parameter_study**
- **steps_per_variable**

Number of steps to take in each dimension of a centered parameter study

**Specification**

*Alias:* deltas_per_variable  
*Argument(s):* INTEGERLIST

**Description**

The `steps_per_variable` keyword allows the user to define the number of steps in each dimension of a centered parameter study. Because they are taken independently, the number of steps can be specified for each.

**Default Behavior**

The user is required to define the number of steps per variable for a centered parameter study. There are no default values.

Steps are taken in the plus and minus directions, and are defined in either actual values (continuous and discrete range) or index offsets (discrete set).

**Examples**

```plaintext
method
  centered_parameter_study
  steps_per_variable = 5 4
  step_vector = 0.4 0.5
```

**model_pointer**

- **Keywords Area**
- **method**
- **centered_parameter_study**
• **model_pointer**

  Identifier for model block to be used by a method

**Topics**

This keyword is related to the topics:

• **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** method use of last model parsed (or use of default model if none parsed)

**Description**

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

**Default Behavior**

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

**Usage Tips**

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

**Examples**

```plaintext
environment
tabular_graphics_data
method_pointer = 'UQ'

method
id_method = 'UQ'
model_pointer = 'SURR'
sampling,
samples = 10
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
                0.1 0.2 0.6
                0.1 0.2 0.6
sample_type lhs
distribution cumulative

model
id_model = 'SURR'
surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
```
6.2. METHOD

```
samples = 121 seed = 5034 rng rnum2
model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'
interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'
responses
  response_functions = 3
  no_gradients
  no_hessians

6.2.69 multidim_parameter_study

- Keywords Area
- method
- multidim_parameter_study

Samples variables on full factorial grid of study points

Topics
This keyword is related to the topics:

- parameter_studies

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>partitions</td>
<td>Dakota Keyword Description</td>
</tr>
</tbody>
</table>

Samples variables on full factorial grid of study points
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>model_pointer</th>
<th>Identifier for model block to be used by a method</th>
</tr>
</thead>
</table>

**Description**

Dakota’s multidimensional parameter study computes response data sets for an n-dimensional grid of points. Each continuous and discrete range variable is partitioned into equally spaced intervals between its upper and lower bounds, each discrete set variable is partitioned into equally spaced index intervals. The partition boundaries in n-dimensional space define a grid of points, and every point is evaluated.

**Default Behavior**

By default, the multidimensional parameter study operates over all types of variables.

**Expected Outputs**

A multidimensional parameter study produces a set of responses for each parameter set that is generated.

**Usage Tips**

Since the initial values from the variables specification will not be used, they need not be specified.

**Examples**

This example is taken from the Users Manual and is a good comparison to the examples on centered parameter study and vector parameter study.

```plaintext
# tested on Dakota 6.0 on 140501
environment
  tabular_data
    tabular_data_file = 'rosen_multidim.dat'

method
  multidim_parameter_study
    partitions = 10 8

model
  single

variables
  continuous_design = 2
    lower_bounds -2.0 -2.0
    upper_bounds 2.0 2.0
    descriptors 'x1' "x2"

interface
  analysis_driver = 'rosenbrock'
    fork

responses
  response_functions = 1
  no_gradients
  no_hessians
```

This example illustrates the full factorial combinations of parameter values created by the multidim_parameter_study. With 10 and 8 partitions, there are actually 11 and 9 values for each variable. This means that $11 \times 9 = 99$ function evaluations will be required.

**See Also**

These keywords may also be of interest:
6.2. **METHOD**

- centered-parameter-study
- list-parameter-study
- vector-parameter-study

**partitions**

- Keywords Area
- method
- multidim-parameter-study
- partitions

Samples variables on full factorial grid of study points

**Topics**

This keyword is related to the topics:

- parameter-studies

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Description**

Dakota’s multidimensional parameter study computes response data sets for an n-dimensional grid of points. Each continuous and discrete range variable is partitioned into equally spaced intervals between its upper and lower bounds, each discrete set variable is partitioned into equally spaced index intervals. The partition boundaries in n-dimensional space define a grid of points, and every point is evaluated.

**Default Behavior**

By default, the multidimensional parameter study operates over all types of variables.

**Expected Outputs**

A multidimensional parameter study produces a set of responses for each parameter set that is generated.

**Usage Tips**

Since the initial values from the variables specification will not be used, they need not be specified.

**Examples**

This example is taken from the Users Manual and is a good comparison to the examples on centered-parameter-study and vector-parameter-study.

```plaintext
# tested on Dakota 6.0 on 140501
environment
  tabular_data
  tabular_data_file = 'rosen_multidim.dat'

method
  multidim_parameter_study
  partitions = 10 8
```
This example illustrates the full factorial combinations of parameter values created by the multidim-parameter-study. With 10 and 8 partitions, there are actually 11 and 9 values for each variable. This means that $11 \times 9 = 99$ function evaluations will be required.

See Also

These keywords may also be of interest:

- centered_parameter_study
- list_parameter_study
- vector_parameter_study

model_pointer

- Keywords Area
- method
- multidim_parameter_study
- model_pointer

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)
6.2. METHOD

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See `block_pointer` for details about pointers.

Examples

evironment
    tabular_graphics_data
    method_pointer = 'UQ'

method
    id_method = 'UQ'
    model_pointer = 'SRR'
    sampling,
        samples = 10
        seed = 98765 rng rnum2
        response_levels = 0.1 0.2 0.6
        0.1 0.2 0.6
        0.1 0.2 0.6
    sample_type lhs
distribution cumulative

model
    id_model = 'SRR'
    surrogate global,
        dace_method_pointer = 'DACE'
polynomial quadratic

method
    id_method = 'DACE'
    model_pointer = 'DACE_M'
    sampling sample_type lhs
        samples = 121 seed = 5034 rng rnum2

model
    id_model = 'DACE_M'
single
    interface_pointer = 'I1'

variables
    uniform_uncertain = 2
        lower_bounds = 0. 0.
        upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
    id_interface = 'I1'
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses
response_functions = 3  
nogradients  
no_hessians

6.2.70  richardson_extrap

- Keywords Area
- method
- richardson_extrap

Estimate order of convergence of a response as model fidelity increases

Specification

Alias: none  
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>estimate_order</td>
<td>Compute the best estimate of the convergence order from three points</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>converge_order</td>
<td>Refine until the estimated convergence order converges</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>converge_qoi</td>
<td>Refine until the response converges</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>refinement_rate</td>
<td>Rate at which the state variables are refined</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>convergence_tolerance</td>
<td>Stopping criterion based on convergence of the objective function or statistics</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>max_iterations</td>
<td>Stopping criterion based on number of iterations</td>
</tr>
</tbody>
</table>
Description

Solution verification procedures estimate the order of convergence of the simulation response data during the course of a refinement study. This branch of methods is new and currently only contains one algorithm: Richardson extrapolation.

Refinement of the model

The model fidelity must be parameterized by one or more continuous state variable(s).

The refinement path is determined from the initial state of the continuous state variables specification in combination with the refinement rate, where each of the state variables is treated as an independent refinement factor and each of the initial state values is repeatedly divided by the refinement rate value to define new discretization states.

Results

Three algorithm options are currently provided:

1. estimate_order
2. converge_order
3. converge_qoi

Stopping Criteria

The method employs the max_iterations and convergence_tolerance method independent controls as stopping criteria.

Theory

In each of these cases, convergence order for a response quantity of interest (QoI) is estimated from

\[ p = \ln \left( \frac{QoI_3 - QoI_2}{QoI_2 - QoI_1} \right) / \ln(r) \]

where \( r \) is the uniform refinement rate specified by refinement_rate.

estimate_order

- Keywords Area
- method
- richardson_extrap
- estimate_order

Compute the best estimate of the convergence order from three points

Specification

Alias: none

Argument(s): none
Description

The `estimate_order` option is the simplest option. For each of the refinement factors, it evaluates three points along the refinement path and uses these results to perform an estimate of the convergence order for each response function.

Converge Order

- Keywords Area
- Method
- Richardson Extrapolation
- Converge Order

Refine until the estimated convergence order converges.

Specification

Alias: none
Argument(s): none

Description

The `converge_order` option is initialized using the `estimate_order` approach, and additional refinements are performed along the refinement path until the convergence order estimates converge (two-norm of the change in response orders is less than the convergence tolerance).

Converge Qoi

- Keywords Area
- Method
- Richardson Extrapolation
- Converge Qoi

Refine until the response converges.

Specification

Alias: none
Argument(s): none

Description

The `converge_qoi` option is similar to the `converge_order` option, except that the convergence criterion is that the two-norm of the response discretization errors (computed from extrapolation) must be less than the convergence tolerance.
6.2. METHOD

refinement_rate

- Keywords Area
- method
- richardson_extrap
- refinement_rate

Rate at which the state variables are refined

Specification

Alias: none

Argument(s): REAL

Default: 2.

Description

Described on parent page

convergence_tolerance

- Keywords Area
- method
- richardson_extrap
- convergence_tolerance

Stopping criterion based on convergence of the objective function or statistics

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4
Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration. For optimization, it is most commonly a relative convergence tolerance for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: must be satisfied for two consecutive iterations
- NPSOL: defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function).
- NL2SOL: See nl2sol

max_iterations

- Keywords Area
- method
- richardson_extrap
- max_iterations

Stopping criterion based on number of iterations

Topics

This keyword is related to the topics:

- method_independent_controls

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt, local_reliability: 25; global{reliability, interval_est, evidence} / efficient_global: 25*n)
6.2. METHOD

Description
The maximum number of iterations (default: 100). See also max_function_evaluations.

model_pointer

- Keywords Area
- method
- richardson_extrap
- model_pointer

Identifier for model block to be used by a method

Topics
This keyword is related to the topics:
- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed (or use of default model if none parsed)

Description
The model_pointer is used to specify which model block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior
If not specified, a Dakota method will use the last model block parsed. If specified, there must be a model block in the Dakota input file that has a corresponding id_model with the same name.

Usage Tips
- When doing advanced analyses that involve using multiple methods and multiple models, defining a model_pointer for each method is imperative.
- See block_pointer for details about pointers.

Examples
environment
   tabular_graphics_data
   method_pointer = 'UQ'

method
   id_method = 'UQ'
   model_pointer = 'SURR'
   sampling,
      samples = 10
   seed = 98765 rng rnum2
   response_levels = 0.1 0.2 0.6
      0.1 0.2 0.6
          0.1 0.2 0.6
model
  id_model = ‘SURR’
  surrogate global,
  dace_method_pointer = ‘DACE’
  polynomial quadratic
method
  id_method = ‘DACE’
  model_pointer = ‘DACE_M’
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2
model
  id_model = ‘DACE_M’
  single
  interface_pointer = ‘I1’
variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = ‘x1’ ‘x2’
interface
  id_interface = ‘I1’
  system asynch evaluation_concurrency = 5
  analysis_driver = ‘text_book’
responses
  response_functions = 3
  no_gradients
  no_hessians

6.3 model

- Keywords Area
- model

Specifies how variables are mapped into a set of responses

Topics

This keyword is related to the topics:

- block

Specification

Alias: none
Argument(s): none
A model is comprised of a mapping from variables, through an interface, to responses.

**Model Group 1** The type of model can be:

1. `single`
2. `nested`
3. `surrogate`

The input file must specify one of these types. If the type is not specified, Dakota will assume a single model.

**Block Pointers and ID**

Each of these model types supports `variables_pointer` and `responses_pointer` strings for identifying the variables and responses specifications used in constructing the model by cross-referencing with `id_variables` and `id_responses` strings from particular variables and responses keyword specifications.

These pointers are valid for each model type since each model contains a set of variables that is mapped into a set of responses – only the specifics of the mapping differ.
Additional pointers are used for each model type for constructing the components of the variable to response mapping. As an environment specification identifies a top-level method and a method specification identifies a model, a model specification identifies variables, responses, and (for some types) interface specifications. This top-down flow specifies all of the object interrelationships.

Examples

The next example displays a surrogate model specification which selects a quadratic polynomial from among the global approximation methods. It uses a pointer to a design of experiments method for generating the data needed for building the global approximation, reuses any old data available for the current approximation region, and employs the first-order multiplicative approach to correcting the approximation each time correction is requested.

```model
id_model = 'M1'
variables_pointer = 'V1'
responses_pointer = 'R1'
surrogate
  global
    polynomial quadratic
dace_method_pointer = 'DACE'
reuse_samples region
correction multiplicative first_order
```

This example demonstrates the use of identifiers and pointers. It provides the optional model independent specifications for model identifier, variables pointer, and responses pointer as well as model dependent specifications for global surrogates (see `global`).

Finally, an advanced nested model example would be

```model
id_model = 'M1'
variables_pointer = 'V1'
responses_pointer = 'R1'
nested
  optional_interface_pointer = 'OI1'
  optional_interface_responses_pointer = 'OIR1'
  sub_method_pointer = 'SM1'
  primary_variable_mapping = 'X' 'Y'
  secondary_variable_mapping = 'mean' 'mean'
  primary_response_mapping = 1.0 0.0 1.3 0.0 0.0 0.0
  secondary_response_mapping = 0.0 0.0 0.0 0.0 1.3 0.0
```

This example also supplies model independent controls for model identifier, variables pointer, and responses pointer and supplies model dependent controls for specifying details of the nested mapping.

6.3.1 id_model

- Keywords Area
- model
- id_model

Give the model block an identifying name, in case of multiple model blocks
6.3. MODEL

Topics
This keyword is related to the topics:

- block_identifier

Specification
Alias: none
Argument(s): STRING
Default: method use of last model parsed

Description
The model identifier string is supplied with id_model and is used to provide a unique identifier string for use within method specifications (refer to any of the keywords: model_pointer in under one of the methods in the method block, for example: model_pointer)
This is used to determine which model the method will run.

See Also
These keywords may also be of interest:

- model_pointer

6.3.2 variables_pointer

- Keywords Area
- model
- variables_pointer

Specify which variables block will be included with this model block

Topics
This keyword is related to the topics:

- block_pointer

Specification
Alias: none
Argument(s): STRING
Default: model use of last variables parsed

Description
The variables_pointer is used to specify which variables block will be used by the model, by cross-referencing with id_variables keyword in the variables block.
See block_pointer for details about pointers.
6.3.3 responses_pointer

- Keywords Area
- model
- responses_pointer

Specify which responses block will be used by this model block.

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none
Argument(s): STRING
Default: model use of last responses parsed

Description

The responses_pointer is used to specify which responses block will be used by the model, by cross-referencing with id_responses keyword in the responses block.

See block_pointer for details about pointers.

6.3.4 hierarchical_tagging

- Keywords Area
- model
- hierarchical_tagging

Enables hierarchical evaluation tagging.

Specification

Alias: none
Argument(s): none
Default: no hierarchical tagging

Description

The hierarchical tagging option is useful for studies involving multiple models with a nested or hierarchical relationship. For example a nested model has a sub-method, which itself likely operates on a sub-model, or a hierarchical approximation involves coordination of low and high fidelity models. Specifying hierarchical_tagging will yield function evaluation identifiers ("tags") composed of the evaluation IDs of the models involved, e.g., outermodel.innermodel.interfaceid = 4.9.2. This communicates the outer contexts to the analysis driver when performing a function evaluation.
6.3. MODEL

Examples

test/dakota_uq_timeseries_ivp_optinterf.in test/dakota_uq_timeseries_sop_optinterf.in

See Also

These keywords may also be of interest:

- file_tag model-nested

6.3.5 single

- Keywords Area
- model
- single

A model with one of each block: variable, interface, and response

Specification

Alias: none
Argument(s): none
Default: N/A (single if no model specification)

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>interface_pointer</td>
<td>Interface block pointer for the single model type</td>
</tr>
</tbody>
</table>

Description

The single model is the simplest model type. It uses a single interface instance to map variables into responses. There is no recursion in this case.

The optional interface_pointer specification identifies the interface block by cross-referencing with the id_interface string input from a particular interface keyword specification. This is only necessary when the input file has multiple interface blocks, and you wish to explicitly point to the desired block. The same logic follows for responses and variables blocks and pointers.

Examples

The example shows a minimal specification for a single model, which is the default model when no models are specified by the user.

```
model
    single
```

This example does not provide any pointer strings and therefore relies on the default behavior of constructing the model with the last variables, interface, and responses specifications parsed.
See Also

These keywords may also be of interest:

- surrogate
- nested

interface_pointer

- Keywords Area
- model
- single
- interface_pointer

Interface block pointer for the single model type

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING
Default: model use of last interface parsed

Description

In the single model case, a single interface is used to map the variables into responses. The optional interface-pointer specification identifies this interface by cross-referencing with the id_interface string input from a particular interface keyword specification.

See block_pointer for details about pointers.

6.3.6 surrogate

- Keywords Area
- model
- surrogate

An empirical model that is created from data or the results of a submodel

Specification

Alias: none

Argument(s): none
Surrogate models are inexpensive approximate models that are intended to capture the salient features of an expensive high-fidelity model. They can be used to explore the variations in response quantities over regions of the parameter space, or they can serve as inexpensive stand-ins for optimization or uncertainty quantification studies (see, for example, the surrogate-based optimization methods, surrogate_based_global and surrogate_based_local).

Surrogate models supported in Dakota are categorized as Data Fitting or Hierarchical, as shown below. Each of these surrogate types provides an approximate representation of a "truth" model which is used to perform the parameter to response mappings. This approximation is built and updated using results from the truth model, called the “training data”.

- Data fits:
  
  Data fitting methods involve construction of an approximation or surrogate model using data (response values, gradients, and Hessians) generated from the original truth model. Data fit methods can be further categorized as local, multipoint, and global approximation techniques, based on the number of points used in generating the data fit.

  1. Local: built from response data from a single point in parameter space
     
     - Taylor series expansion: taylor_series
Training data consists of a single point, plus gradient and Hessian information.

2. Multipoint: built from two or more points in parameter space, often involving the current and previous iterates of a minimization algorithm.
   - TANA-3: tana
     Training Data comes from a few previously evaluated points

3. Global full space response surface methods:
   - Polynomial regression: polynomial
   - Gaussian process (Kriging): gaussian_process
   - Artificial neutral network: neural_network
   - MARS: mars
   - Radial Basis Functions: radial_basis
   - Orthogonal polynomials (only supported in PCE/SC for now): polynomial_chaos and stoch_-collocation

Training data is generated using either a design of experiments method applied to the truth model (specified by dace_method_pointer), or from saved data (specified by reuse_points) in a restart database, or an import file.

- Multifidelity/hierarchical:
  Multifidelity modeling involves the use of a low-fidelity physics-based model as a surrogate for the original high-fidelity model. The low-fidelity model typically involves a coarser mesh, looser convergence tolerances, reduced element order, or omitted physics.
  
  See hierarchical.

The global and hierarchical surrogates have a correction feature in order to improve the local accuracy of the surrogate models. The correction factors force the surrogate models to match the true function values and possibly true function derivatives at the center point of each trust region. Details can be found on global correction or hierarchical correction.

Theory

Surrogate models are used extensively in the surrogate-based optimization and least squares methods, in which the goals are to reduce expense by minimizing the number of truth function evaluations and to smooth out noisy data with a global data fit. However, the use of surrogate models is not restricted to optimization techniques; uncertainty quantification and optimization under uncertainty methods are other primary users.

Data Fit Surrogate Models

A surrogate of the \{data fit\} type is a non-physics-based approximation typically involving interpolation or regression of a set of data generated from the original model. Data fit surrogates can be further characterized by the number of data points used in the fit, where a local approximation (e.g., first or second-order Taylor series) uses data from a single point, a multipoint approximation (e.g., two-point exponential approximations (TPEA) or two-point adaptive nonlinearity approximations (TANA)) uses a small number of data points often drawn from the previous iterates of a particular algorithm, and a global approximation (e.g., polynomial response surfaces, kriging/ gaussian_process, neural networks, radial basis functions, splines) uses a set of data points distributed over the domain of interest, often generated using a design of computer experiments.

Dakota contains several types of surface fitting methods that can be used with optimization and uncertainty quantification methods and strategies such as surrogate-based optimization and optimization under uncertainty. These are: polynomial models (linear, quadratic, and cubic), first-order Taylor series expansion, kriging spatial interpolation, artificial neural networks, multivariate adaptive regression splines, radial basis functions, and moving
least squares. With the exception of Taylor series methods, all of the above methods listed in the previous sentence are accessed in Dakota through the Surfpack library. All of these surface fitting methods can be applied to problems having an arbitrary number of design parameters. However, surface fitting methods usually are practical only for problems where there are a small number of parameters (e.g., a maximum of somewhere in the range of 30-50 design parameters). The mathematical models created by surface fitting methods have a variety of names in the engineering community. These include surrogate models, meta-models, approximation models, and response surfaces. For this manual, the terms surface fit model and surrogate model are used.

The data fitting methods in Dakota include software developed by Sandia researchers and by various researchers in the academic community.

**Multifidelity Surrogate Models**

A second type of surrogate is the \{model hierarchy\} type (also called multifidelity, variable fidelity, variable complexity, etc.). In this case, a model that is still physics-based but is of lower fidelity (e.g., coarser discretization, reduced element order, looser convergence tolerances, omitted physics) is used as the surrogate in place of the high-fidelity model. For example, an inviscid, incompressible Euler CFD model on a coarse discretization could be used as a low-fidelity surrogate for a high-fidelity Navier-Stokes model on a fine discretization.

**Surrogate Model Selection**

This section offers some guidance on choosing from among the available surrogate model types.

- For Surrogate Based Local Optimization, using the surrogate\_based\_local method with a trust region:
  
  using the keywords:

  1. surrogate local taylor\_series or
  2. surrogate multipoint tana

  will probably work best.

  If for some reason you wish or need to use a global surrogate (not recommended) then the best of these options is likely to be either:

  1. surrogate global gaussian\_process surfpack or
  2. surrogate global moving\_least\_squares.

- For Efficient Global Optimization (EGO), the efficient\_global method:

  the default surrogate is: gaussian\_process surfpack which is likely to find a more optimal value and/or require fewer true function evaluations than the alternative, gaussian\_process dakota. However, the surfpack will likely take more time to build than the dakota version. Note that currently the use\_derivatives keyword is not recommended for use with EGO based methods.

- For EGO based global interval estimation, the global\_interval\_est ego method:

  the default gaussian\_process surfpack will likely work better than the alternative gaussian\_process dakota.

- For Efficient Global Reliability Analysis (EGRA), the global\_reliability method:

  the surfpack and dakota versions of the gaussian process tend to give similar answers with the dakota version tending to use fewer true function evaluations. Since this is based on EGO, it is likely that the default surfpack is more accurate, although this has not been rigorously demonstrated.

- For EGO based Dempster-Shafer Theory of Evidence, i.e. the global\_evidence ego method, the default gaussian\_process surfpack often use significantly fewer true function evaluations than the alternative gaussian\_process dakota.

- When using a global surrogate to extrapolate, any of the surrogates:
– gaussian_process_surfpack
– polynomial quadratic
– polynomial cubic

are recommended.

• When there is over roughly two or three thousand data points and you wish to interpolate (or approximately interpolate) then a Taylor series, Radial Basis Function Network, or Moving Least Squares fit is recommended. The only reason that the gaussian_process_surfpack is not recommended is that it can take a considerable amount of time to construct when the number of data points is very large. Use of the third party MARS package included in Dakota is generally discouraged.

• In other situations that call for a global surrogate, the gaussian_process_surfpack is generally recommended. The use_derivatives keyword will only be useful if accurate and inexpensive derivatives are available. Finite difference derivatives are disqualified on both counts. However, derivatives generated by analytical, automatic differentiation, or continuous adjoint techniques can be appropriate. Currently, first order derivatives, i.e. gradients, are the highest order derivatives that can be used to construct the gaussian_process_surfpack model; Hessians will not be used even if they are available.

See Also
These keywords may also be of interest:

• single
• nested

id_surrogates

• Keywords Area
• model
• surrogate
• id_surrogates

Identifies the subset of the response functions by number that are to be approximated (the default is all functions).

Specification
Alias: none
Argument(s): INTEGERLIST
Default: All response functions are approximated

Description
In the surrogate model case, the specification first allows a mixture of surrogate and actual response mappings through the use of the optional id_surrogates specification. This identifies the subset of the response functions by number that are to be approximated (the default is all functions). The valid response function identifiers range from 1 through the total number of response functions (see response_functions).
6.3. MODEL

global
- Keywords Area
- model
- surrogate
- global

Select a surrogate model with global support

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required(Choose One)</strong></td>
<td>Group 1</td>
<td>gaussian_process</td>
<td>Gaussian Process surrogate model</td>
</tr>
<tr>
<td>mars</td>
<td></td>
<td></td>
<td>Multivariate Adaptive Regression Spline (MARS)</td>
</tr>
<tr>
<td>moving_least-squares</td>
<td></td>
<td></td>
<td>Moving Least Squares surrogate models</td>
</tr>
<tr>
<td>neural_network</td>
<td></td>
<td></td>
<td>Artificial neural network model</td>
</tr>
<tr>
<td>radial_basis</td>
<td></td>
<td></td>
<td>Radial basis function (RBF) model</td>
</tr>
<tr>
<td>polynomial</td>
<td></td>
<td></td>
<td>Polynomial surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>domain_decomposition</td>
<td>Piecewise Domain Decomposition for Global Surrogate Models</td>
</tr>
<tr>
<td>Optional <em>(Choose One)</em></td>
<td>Group 2</td>
<td>total_points</td>
<td>Specified number of training points</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>--------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>minimum_points</td>
<td>Construct surrogate with minimum number of points</td>
</tr>
<tr>
<td></td>
<td></td>
<td>recommended_points</td>
<td>Construct surrogate with recommended number of points</td>
</tr>
<tr>
<td>Optional <em>(Choose One)</em></td>
<td>Group 3</td>
<td>dace_method_pointer</td>
<td>Specify a method to gather training data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>actual_model_pointer</td>
<td>A surrogate model pointer that guides a method to whether it should use a surrogate model or compute truth function evaluations</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>reuse_points</td>
<td>Surrogate model training data reuse control</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>import_build_points_file</td>
<td>File containing points you wish to use to build a surrogate</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>export_approx_points_file</td>
<td>Output file for evaluations of a surrogate model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>use_derivatives</td>
<td>Use derivative data to construct surrogate models</td>
</tr>
</tbody>
</table>
Optional correction

Optional metrics

Optional import challenge -
points_file

Correction approaches for surrogate models
Compute surrogate quality metrics
Datafile of points to assess surrogate quality

Description

The global surrogate model requires specification of one of the following approximation types:

1. Polynomial
2. Gaussian process (Kriging interpolation)
3. Layered perceptron artificial neural network approximation
4. MARS
5. Moving least squares
6. Radial basis function
7. Voronoi Piecewise Surrogate (VPS)

All these approximations are implemented in SurfPack[36], except for VPS. In addition, a second version of Gaussian process is implemented directly in Dakota.

Training Data

Training data can be taken from prior runs, stored in a datafile, or by running a Design of Experiments method. The keywords listed below are used to determine how to collect training data:

- dace_method_pointer
- reuse_points
- import_points_file
- use_derivatives

The source of training data is determined by the contents of a provided import_-points_file, whether reuse_points and use_derivatives are specified, and the contents of the method block specified by dace_method_pointer. use_derivatives is a special case, the other keywords are discussed below.

The number of training data points used in building a global approximation is determined by specifying one of three point counts:

1. minimum_points: minimum required or minimum "reasonable" amount of training data. Defaults to d+1 for d input dimensions for most models, e.g., polynomials override to the number of coefficients required to estimate the requested order.
2. recommended_points: recommended number of training data, (this is the default option, if none of the keywords is specified). Defaults to 5*d, except for polynomials where it’s equal to the minimum.
3. **total_points**: specify the number of training data points. However, if the **total_points** value is less than the default **minimum_points** value, the **minimum_points** value is used.

The sources of training data depend on the number of training points, \(N_{tp}\), the number of points in the import file, \(N_{if}\), and the value of **reuse_points**.

- If there is no import file, all training data come from the DACE method
- If there is an import file, all \(N_{if}\) points from the file are used, and the remaining \(N_{tp} - N_{if}\) points come from the DACE method
- If there is an import file and **reuse_points** is:
  - **none** - all \(N_{tp}\) points from DACE method
  - **region** - only the points within a trust region are taken from the import file, and all remaining points are from the DACE method.
  - **all** - (Default) all \(N_{if}\) points from the file are used, and the remaining \(N_{tp} - N_{if}\) points come from the DACE method

**Surrogate Correction**
A correction model can be added to the constructed surrogate in order to better match the training data. The specified correction method will be applied to the surrogate, and then the corrected surrogate model is used by the method.

Finally, the quality of the surrogate can be tested using the **metrics** and **challenge_points_file** keywords.

**Theory**

Global methods, also referred to as response surface methods, involve many points spread over the parameter ranges of interest. These surface fitting methods work in conjunction with the sampling methods and design of experiments methods.

**Procedures for Surface Fitting**
The surface fitting process consists of three steps:

1. selection of a set of design points
2. evaluation of the true response quantities (e.g., from a user-supplied simulation code) at these design points,
3. using the response data to solve for the unknown coefficients (e.g., polynomial coefficients, neural network weights, kriging correlation factors) in the surface fit model.

In cases where there is more than one response quantity (e.g., an objective function plus one or more constraints), then a separate surface is built for each response quantity. Currently, the surface fit models are built using only 0th-order information (function values only), although extensions to using higher-order information (gradients and Hessians) are possible.

Each surface fitting method employs a different numerical method for computing its internal coefficients. For example, the polynomial surface uses a least-squares approach that employs a singular value decomposition to compute the polynomial coefficients, whereas the kriging surface uses Maximum Likelihood Estimation to compute its correlation coefficients. More information on the numerical methods used in the surface fitting codes is provided in the Dakota Developers Manual.
See Also
These keywords may also be of interest:

- local
- hierarchical
- multipoint

**gaussian.process**

- Keywords Area
- model
- surrogate
- global
- `gaussian_process`

Gaussian Process surrogate model

**Specification**

**Alias:** kriging

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>dakota</td>
<td>Select the built in Gaussian Process surrogate</td>
</tr>
<tr>
<td>surfpack</td>
<td></td>
<td>surfpack</td>
<td>Use the Surfpack version of Gaussian Process surrogates</td>
</tr>
</tbody>
</table>

**Description**

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the `surfpack` keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the `dakota` version is deprecated and intended to be removed in a future release.**

**dakota**

- Keywords Area
- model
- surrogate
- global
- `gaussian_process`
- `dakota`

Select the built in Gaussian Process surrogate
CHAPTER 6. KEYWORDS AREA

 Specification

 Alias: none
 Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>point_selection</td>
<td>Enable greedy selection of well-spaced build points</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>trend</td>
<td>Choose a trend function for a Gaussian process surrogate</td>
</tr>
</tbody>
</table>

 Description

 A second version of GP surrogates was available in prior versions of Dakota. For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.

 Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota’s. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

 More details on the gaussian_process dakota model can be found in[58].

 Dakota’s GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

 This differs from the point_selection option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

 point_selection

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- dakota
- point_selection

 Enable greedy selection of well-spaced build points
Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): none
Default: no point selection

Description
The Dakota Gaussian Process model has a point_selection option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

trend

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- dakota
- trend

Choose a trend function for a Gaussian process surrogate

Specification
Alias: none
Argument(s): none
Default: reduced_quadratic

| Required/- | Description of Group 1 | Dakota Keyword | Dakota Keyword Description |
| Optional | | constant | Description Constant trend function |

<table>
<thead>
<tr>
<th>Required (Choose One)</th>
<th>Required/-</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optional</td>
<td>Group 1</td>
<td>constant</td>
<td>Description Constant trend function</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>linear</strong></td>
<td>Use a linear polynomial or trend function</td>
<td></td>
</tr>
<tr>
<td><strong>reduced_quadratic</strong></td>
<td>Quadratic polynomials - main effects only</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The only trend functions that are currently supported are polynomials.

The trend function is selected using the `trend` keyword, with options `constant`, `linear`, or `reduced_quadratic`. The `reduced_quadratic` trend function includes the main effects, but not mixed/interaction terms. The Surfpack GP (See surfpack) has the additional option of (a full) quadratic.

**constant**

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- dakota
- trend
- constant

Constant trend function

**Specification**

Alias: none

Argument(s): none

**Description**

See parent page

**linear**

- Keywords Area
- model
- surrogate
- global
Use a linear polynomial or trend function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent page

**reduced_quadratic**

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- dakota
- trend
- reduced_quadratic

Quadratic polynomials - main effects only

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

In 2 or more dimensions, this polynomial omits the interaction, or mixed, terms.
surfpack

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack

Use the Surfpack version of Gaussian Process surrogates

### Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional trend</td>
<td></td>
<td>trend</td>
<td>Choose a trend function for a Gaussian process surrogate</td>
</tr>
<tr>
<td>Optional optimization_method</td>
<td></td>
<td>optimization_method</td>
<td>Change the optimization method used to compute hyperparameters</td>
</tr>
<tr>
<td>Optional max_trials</td>
<td></td>
<td>max_trials</td>
<td>Max number of likelihood function evaluations</td>
</tr>
<tr>
<td>Optional nugget</td>
<td></td>
<td>nugget</td>
<td>Specify a nugget to handle ill-conditioning</td>
</tr>
<tr>
<td>Optional Group 1 find Nugget</td>
<td></td>
<td>find_nugget</td>
<td>Have Surfpack compute a nugget to handle ill-conditioning</td>
</tr>
<tr>
<td>Optional correlation_lengths</td>
<td></td>
<td>correlation_lengths</td>
<td>Specify the correlation lengths for the Gaussian process</td>
</tr>
</tbody>
</table>


6.3. MODEL

| Optional | export_model | Exports surrogate model in user-selected format |

**Description**

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. **Optimization methods**:
   
   Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See `optimization_method`.
   
   The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. **Trend Function**:
   
   The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See `trend`.

3. **Correlation Lengths**:
   
   Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See `correlation_lengths`.

4. **Ill-conditioning**
   
   One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.
   
   Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.
   
   The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. **Gradient Enhanced Kriging (GEK)**
   
   The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).
   
   See notes in the Theory section.

**Theory**

**Gradient Enhanced Kriging**

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use
derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK’s correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging’s correlation matrix (which is a small sub-matrix within GEK’s correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK’s correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging’s correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK’s correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK’s correlation matrix.

trend

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack
- trend

Choose a trend function for a Gaussian process surrogate

**Specification**

Alias: none

Argument(s): none

Default: reduced_quadratic

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required (Choose One)</strong></td>
<td>Group 1</td>
<td>constant</td>
<td>Constant trend function</td>
</tr>
</tbody>
</table>

linear Use a linear polynomial or trend function
6.3. MODEL

<table>
<thead>
<tr>
<th></th>
<th>reduced_quadratic</th>
<th>Quadratic polynomials - main effects only</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadratic</td>
<td></td>
<td>Use a quadratic polynomial or trend function</td>
</tr>
</tbody>
</table>

**Description**

The only trend functions that are currently supported are polynomials.

The trend function is selected using the `trend` keyword, with options `constant`, `linear`, or `reduced_quadratic`. The `reduced_quadratic` trend function includes the main effects, but not mixed/interaction terms. The Surfpack GP (See surfpack) has the additional option of (a full) quadratic.

**constant**

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack
- trend
- constant

Constant trend function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent page

**linear**

- Keywords Area
- model
- surrogate
- global
Use a linear polynomial or trend function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent page

**reduced_quadratic**

- **Keywords Area**

- **model**

- **surrogate**

- **global**

- **gaussian_process**

- **surfpack**

- **trend**

- **reduced_quadratic**

Quadratic polynomials - main effects only

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

In 2 or more dimensions, this polynomial omits the interaction, or mixed, terms.
6.3. MODEL

quadratic
- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack
- trend
- quadratic

Use a quadratic polynomial or trend function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent page

**optimization_method**

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack
- optimization_method

Change the optimization method used to compute hyperparameters

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** global
Description
Select the optimization method to compute hyperparameters of the Gaussian Process by specifying one of these arguments:

- **global** (default) - DIRECT method
- **local** - CONMIN method
- **sampling** - generates several random guesses and picks the candidate with greatest likelihood
- **none** - no optimization, pick the center of the feasible region

The **none** option, and the starting location of the **local** optimization, default to the center, in log(correlation length) scale, of the of feasible region.
Surfpack picks a small feasible region of correlation parameters.
Note that we have found the **global** optimization method to be the most robust.

**max_trials**

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack
- **max_trials**

Max number of likelihood function evaluations

Specification
Alias: none
Argument(s): INTEGER

Description
See parent page

**nugget**

- Keywords Area
- model
- surrogate
- global
- gaussian_process
6.3. MODEL

- surfpack
- nugget

Specify a nugget to handle ill-conditioning

**Specification**

**Alias:** none

**Argument(s):** REAL

**Default:** None

**Description**

By default, the Surfpack GP handles ill-conditioning and does not use a nugget. If the user wishes to specify a nugget, there are two approaches.

- The user can specify the value of a nugget with `nugget`.
- Have Surfpack find the optimal value of the nugget. This is specified by `find_nugget`. There are two options for `find_nugget`.
  
  - `find_nugget = 1`: assume that the reciprocal condition number of the correlation matrix R, rcondR, is zero and calculate the nugget needed to make the worst case of R not ill-conditioned.
  
  - `find_nugget = 2`: calculate rcondR, which requires a Cholesky factorization. If rcondR indicates that R is not ill-conditioned, then kriging uses the Cholesky factorization. Otherwise, if rcondR says R is ill conditioned, then kriging will calculate the nugget needed to make the worst case of R not ill-conditioned.

`find_nugget = 1` and `2` are similar, the second option just takes more computation (the initial Cholesky factorization) for larger problems.

**find_nugget**

- **Keywords Area**
- model
- surrogate
- global
- gaussian_process
- surfpack
- find_nugget

Have Surfpack compute a nugget to handle ill-conditioning

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** None
CHAPTER 6. KEYWORDS AREA

Description

By default, the Surfpack GP handles ill-conditioning and does not use a nugget. If the user wishes to specify a nugget, there are two approaches.

- The user can specify the value of a nugget with `nugget`.
- Have Surfpack find the optimal value of the nugget. This is specified by `find_nugget`. There are two options for `find_nugget`.
  - `find_nugget = 1`: assume that the reciprocal condition number of the correlation matrix R, rcondR, is zero and calculate the nugget needed to make the worst case of R not ill-conditioned.
  - `find_nugget = 2`: calculate rcondR, which requires a Cholesky factorization. If rcondR indicates that R is not ill-conditioned, then kriging uses the Cholesky factorization. Otherwise, if rcondR says R is ill conditioned, then kriging will calculate the nugget needed to make the worst case of R not ill conditioned.

`find_nugget = 1` and 2 are similar, the second option just takes more computation (the initial Cholesky factorization) for larger problems.

correlation_lengths

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **gaussian_process**
- **surfpack**
- **correlation_lengths**

Specify the correlation lengths for the Gaussian process

Specification

Alias: none

- **Argument(s):** `REALLIST`
- **Default:** internally computed `correlation_lengths`

Description

Directly specify `correlation_lengths` as a list of N real numbers where N is the number of input dimensions.
export_model

• Keywords Area
• model
• surrogate
• global
• gaussian_process
• surfpack
• export_model

Exports surrogate model in user-selected format

Topics

This keyword is related to the topics:

• surrogate_models

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>filename_prefix</td>
<td>formats</td>
<td>User-customizable portion of exported model filenames</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td>Formats for surrogate model export</td>
</tr>
</tbody>
</table>

Description

Export the surrogate for later evaluation using the surfpack executable (bin/surfpack) or a user-developed tool. Export format is controlled using the formats specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

• text_archive - Plain-text, machine-readable archive for use with the surfpack executable
• binary_archive - Binary, machine-readable archive for use with the surfpack executable
• algebraic_file - Plain-text, human-readable file intended for use with user-created tools; not compatible with the surfpack executable
• algebraic_console - Print the model in algebraic format to the screen; not compatible with the surfpack executable
Most global surrogates can be exported in all four formats. These include:

- Gaussian process (keyword `gaussian_process surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported. In addition, Dakota cannot import models that have been exported. They are strictly for use with external tools.

**Default Behavior**
No export.

**Expected Output**
Output depends on selected format; see the `formats` specification.

**Additional Discussion**
The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

### filename_prefix

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack
- export_model
- filename_prefix

User-customizable portion of exported model filenames

**Topics**
This keyword is related to the topics:

- `surrogate_models`

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** exported_surrogate
6.3. MODEL

Description

When a file-based export of a surrogate model is specified, Dakota writes one file per response, per requested format. The files are named using the pattern \{prefix\}.\{response_descriptor\}.\{extension\}.

The response_descriptor portion of the pattern is filled in using the response descriptors provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format. The filename_prefix keyword is used to supply the prefix portion of the pattern.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns my_surrogate.\{response_descriptor\}.alg (for the algebraic files) and my_surrogate.\{response_descriptor\}.bss (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

Formats for surrogate model export

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): none
### Description
Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

**text_archive**
- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **gaussian_process**
- **surfpack**
- **export_model**
- **formats**
- **text_archive**

Export surrogate model to a plain-text archive file

### Topics
This keyword is related to the topics:
- **surrogate_models**

### Specification
**Alias:** none  
**Argument(s):** none
6.3. MODEL

Description

After the surrogate model has been built, Dakota will export it to a plain-text archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern \{prefix\}.{response_descriptor}.sps, in which ‘sps’ stands for Surfpack Surrogate. See filename_prefix for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

• binary_archive
  • Keywords Area
  • model
  • surrogate
  • global
  • gaussian_process
  • surfpack
  • export_model
  • formats
  • binary_archive

Export surrogate model to a binary archive file

Topics

This keyword is related to the topics:

• surrogate_models

Specification

Alias: none
  Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a binary archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern \{prefix\}.{response_descriptor}.bsps, in which ‘bsps’ stands for Binary Surfpack Surrogate. See filename_prefix for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.
algebraic_file

- Keywords Area
- model
- surrogate
- global
- gaussian_process
- surfpack
- export_model
- formats
- algebraic_file

Export surrogate model in algebraic format to a file

Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): none

Description
After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern \{prefix\}.\{response_descriptor\}.alg. See filename_prefix for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output
The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when algebraic_console is specified.
6.3. MODEL

- gaussian_process
- surfpack
- export_model
- formats
- algebraic_console

Export surrogate model in algebraic format to the console

Topics
This keyword is related to the topics:
- surrogate_models

Specification

Alias: none
Argument(s): none

Description
After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the -o option) in a human-readable "algebraic" format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output
The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when algebraic_file is specified. Use of algebraic_file is preferred over algebraic_console, which exists largely to provide a measure of backward compatibility.

mars

- Keywords Area
- model
- surrogate
- global
- mars

Multivariate Adaptive Regression Spline (MARS)

Specification

Alias: none
Argument(s): none
Description

This surface fitting method uses multivariate adaptive regression splines from the MARS3.5 package[27] developed at Stanford University.

The MARS reference material does not indicate the minimum number of data points that are needed to create a MARS surface model. However, in practice it has been found that at least $n_{\text{quad}}$, and sometimes as many as 2 to 4 times $n_{\text{quad}}$, data points are needed to keep the MARS software from terminating. Provided that sufficient data samples can be obtained, MARS surface models can be useful in SBO and OUU applications, as well as in the prediction of global trends throughout the parameter space.

Theory

The form of the MARS model is based on the following expression:

$$\hat{f}(x) = \sum_{m=1}^{M} a_m B_m(x)$$

where the $a_m$ are the coefficients of the truncated power basis functions $B_m$, and $M$ is the number of basis functions. The MARS software partitions the parameter space into subregions, and then applies forward and backward regression methods to create a local surface model in each subregion. The result is that each subregion contains its own basis functions and coefficients, and the subregions are joined together to produce a smooth, $C^2$-continuous surface model.

MARS is a nonparametric surface fitting method and can represent complex multimodal data trends. The regression component of MARS generates a surface model that is not guaranteed to pass through all of the response data values. Thus, like the quadratic polynomial model, it provides some smoothing of the data.

required/
optional

<table>
<thead>
<tr>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_bases</td>
<td>max_bases</td>
<td>Maximum number of MARS bases</td>
</tr>
<tr>
<td>interpolation</td>
<td>interpolation</td>
<td>MARS model interpolation type</td>
</tr>
<tr>
<td>export_model</td>
<td>export_model</td>
<td>Exports surrogate model in user-selected format</td>
</tr>
</tbody>
</table>

- **max_bases**
  - Keywords Area
  - model
  - surrogate
  - global
  - mars
6.3. MODEL

- max_bases

  Maximum number of MARS bases

**Specification**

**Alias:** none

  **Argument(s):** INTEGER

**Description**

The maximum number of basis functions allowed in the MARS approximation model.

**interpolation**

  **Keywords Area**

  - model
  - surrogate
  - global
  - mars
  - interpolation

  MARS model interpolation type

**Specification**

**Alias:** none

  **Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td></td>
<td>linear</td>
<td>Linear interpolation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cubic</td>
<td>Cubic interpolation</td>
</tr>
</tbody>
</table>

**Description**

The MARS model interpolation type: linear or cubic.

**linear**

  **Keywords Area**

  - model
  - surrogate
  - global
• mars
• interpolation
• linear
Linear interpolation

**Specification**

Alias: none
Argument(s): none

**Description**

Use linear interpolation in the MARS model.

cubic
• Keywords Area
• model
• surrogate
• global
• mars
• interpolation
• cubic
Cubic interpolation

**Specification**

Alias: none
Argument(s): none

**Description**

Use cubic interpolation in the MARS model.

elexport_model
• Keywords Area
• model
• surrogate
• global
• mars
• export_model
Exports surrogate model in user-selected format
6.3. **MODEL**

**Topics**

This keyword is related to the topics:

- surrogate_models

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>filename_prefix</td>
<td>formats</td>
<td>User-customizable portion of exported model filenames</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>formats</td>
<td>Formats for surrogate model export</td>
</tr>
</tbody>
</table>

**Description**

Export the surrogate for later evaluation using the surfpack executable (bin/surfpack) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- **text_archive** - Plain-text, machine-readable archive for use with the surfpack executable
- **binary_archive** - Binary, machine-readable archive for use with the surfpack executable
- **algebraic_file** - Plain-text, human-readable file intended for use with user-created tools; not compatible with the surfpack executable
- **algebraic_console** - Print the model in algebraic format to the screen; not compatible with the surfpack executable

Most global surrogates can be exported in all four formats. These include:

- Gaussian process (keyword `gaussian_process surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Funtions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used. Currently, no other surrogate models can be exported. In addition, Dakota cannot import models that have been exported. They are strictly for use with external tools.

**Default Behavior**

No export.

**Expected Output**
Output depends on selected format; see the formats specification.

Additional Discussion
The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

filename_prefix

- Keywords Area
- model
- surrogate
- global
- mars
- export_model
- filename_prefix

User-customizable portion of exported model filenames

Topics
This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description
When a file-based export of a surrogate model is specified, Dakota writes one file per response, per requested format. The files are named using the pattern \{prefix\}.\{response_descriptor\}.\{extension\}.

The response_descriptor portion of the pattern is filled in using the response descriptors provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format. The filename_prefix keyword is used to supply the prefix portion of the pattern.

Examples
This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns my_surrogate.\{response_descriptor\}.alg (for the algebraic files) and my_surrogate.\{response_descriptor\}.bsps (for the binary files).

```plaintext
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```
6.3. MODEL

Formats

- Keywords Area
- model
- surrogate
- global
- mars
- export_model
- formats

Formats for surrogate model export

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>text_archive</td>
<td>text_archive</td>
<td>Export surrogate model to a plain-text archive file</td>
</tr>
<tr>
<td>Optional</td>
<td>binary_archive</td>
<td>binary_archive</td>
<td>Export surrogate model to a binary archive file</td>
</tr>
</tbody>
</table>

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See export_model and the entries for the format selection keywords for further information.

text_archive

- Keywords Area
- model
- surrogate
- global
- mars
Export surrogate model to a plain-text archive file

**Topics**

This keyword is related to the topics:

- surrogate_models

**Specification**

Alias: none  
Argument(s): none

**Description**

After the surrogate model has been built, Dakota will export it to a plain-text archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern `{prefix}.{response_descriptor}.sps`, in which ’sps’ stands for Surfpack Surrogate. See `filename_prefix` for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

Export surrogate model to a binary archive file

**Topics**

This keyword is related to the topics:

- surrogate_models
6.3. MODEL

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a binary archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern \{prefix\}.\{response_descriptor\}.bsps, in which 'bsps' stands for Binary Surfpack Surrogate. See filename_prefix for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

**moving_least_squares**

- Keywords Area
- model
- surrogate
- global
- moving_least_squares

Moving Least Squares surrogate models

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>basis_order</td>
<td>Polynomial order for the MLS bases</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>weight_function</td>
<td>Selects the weight function for the MLS model</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>export_model</td>
<td>Exports surrogate model in user-selected format</td>
</tr>
</tbody>
</table>

Description

Moving least squares is a further generalization of weighted least squares where the weighting is "moved" or recalculated for every new point where a prediction is desired[64].
The implementation of moving least squares is still under development. It tends to work well in trust region optimization methods where the surrogate model is constructed in a constrained region over a few points. The present implementation may not work as well globally.

Theory
Moving Least Squares can be considered a more specialized version of linear regression models. In linear regression, one usually attempts to minimize the sum of the squared residuals, where the residual is defined as the difference between the surrogate model and the true model at a fixed number of points.

In weighted least squares, the residual terms are weighted so the determination of the optimal coefficients governing the polynomial regression function, denoted by \( f(x) \), are obtained by minimizing the weighted sum of squares at \( N \) data points:

\[
\sum_{n=1}^{N} w_n \left( \| \hat{f}(x_n) - f(x_n) \| \right)
\]

basis_order
- Keywords Area
- model
- surrogate
- global
- moving_least_squares
- basis_order
  Polynomial order for the MLS bases

Specification
Alias: poly_order
Argument(s): INTEGER

Description
The polynomial order for the moving least squares basis function (default = 2).

weight_function
- Keywords Area
- model
- surrogate
- global
- moving_least_squares
- weight_function
  Selects the weight function for the MLS model
6.3. MODEL

Specification

Alias: none
Argument(s): INTEGER

Description

The weight function decays as a function of distance from the training data. Specify one of:

- 1 (default): exponential decay in weight function; once differentiable MLS model
- 2: twice differentiable MLS model
- 3: three times differentiable MLS model

export_model

- Keywords Area
- model
- surrogate
- global
- moving_least_squares
- export_model

Exports surrogate model in user-selected format

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>filename_prefix</td>
<td>filename_prefix</td>
<td>User-customizable portion of exported model filenames</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

Export the surrogate for later evaluation using the surfpack executable (bin/surfpack) or a user-developed tool. Export format is controlled using the formats specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- **text_archive** - Plain-text, machine-readable archive for use with the surfpack executable
- **binary_archive** - Binary, machine-readable archive for use with the surfpack executable
- **algebraic_file** - Plain-text, human-readable file intended for use with user-created tools; not compatible with the surfpack executable
- **algebraic_console** - Print the model in algebraic format to the screen; not compatible with the surfpack executable

Most global surrogates can be exported in all four formats. These include:

- Gaussian process (keyword `gaussian_process surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only **text_archive** and **binary_archive** formats may be used.

Currently, no other surrogate models can be exported. In addition, Dakota cannot import models that have been exported. They are strictly for use with external tools.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the formats specification.

Additional Discussion

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

- **filename_prefix**
  - Keywords Area
  - model
  - surrogate
  - global
  - moving_least_squares
  - export_model
  - filename_prefix

User-customizable portion of exported model filenames
6.3. MODEL

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): STRING
Default: exported_surrogate

Description

When a file-based export of a surrogate model is specified, Dakota writes one file per response, per requested format. The files are named using the pattern \{prefix\}.\{response_descriptor\}.\{extension\}.

The response_descriptor portion of the pattern is filled in using the response descriptors provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format. The filename_prefix keyword is used to supply the prefix portion of the pattern.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns my_surrogate.\{response_descriptor\}.alg (for the algebraic files) and my_surrogate.\{response_descriptor\}.bsps (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

Formats

- Keywords Area
- model
- surrogate
- global
- moving_least_squares
- export_model
- formats

Formats for surrogate model export
Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>text_archive</td>
<td>Export surrogate model to a plain-text archive file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>binary_archive</td>
<td>Export surrogate model to a binary archive file</td>
</tr>
</tbody>
</table>

Description
Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See export_model and the entries for the format selection keywords for further information.

text_archive
- Keywords Area
- model
- surrogate
- global
- moving_least_squares
- export_model
- formats
- text_archive

Export surrogate model to a plain-text archive file

Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): none
6.3. **MODEL**

### Description

After the surrogate model has been built, Dakota will export it to a plain-text archival file suitable only for use with the surfpack executable (`bin/surfpack`). The file is named using the pattern `{prefix}.{response_descriptor}.sps`, in which ‘sps’ stands for Surfpack Surrogate. See `filename_prefix` for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

**binary_archive**

- Keywords Area
- model
- surrogate
- global
- moving_least_squares
- export_model
- formats
- binary_archive

Export surrogate model to a binary archive file

### Topics

This keyword is related to the topics:

- surrogate_models

### Specification

**Alias:** none

**Argument(s):** none

### Description

After the surrogate model has been built, Dakota will export it to a binary archival file suitable only for use with the surfpack executable (`bin/surfpack`). The file is named using the pattern `{prefix}.{response_descriptor}.bsps`, in which ‘bsps’ stands for Binary Surfpack Surrogate. See `filename_prefix` for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.
neural_network

- Keywords Area
- model
- surrogate
- global
- neural_network

Artificial neural network model

**Specification**

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>max_nodes</td>
<td>Maximum number of hidden layer nodes</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>range</td>
<td>Range for neural network random weights</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>random_weight</td>
<td>(Inactive) Random weight control</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>export_model</td>
<td>Exports surrogate model in user-selected format</td>
</tr>
</tbody>
</table>

**Description**

Dakota’s artificial neural network surrogate is a stochastic layered perceptron network, with a single hidden layer. Weights for the input layer are chosen randomly, while those in the hidden layer are estimated from data using a variant of the Zimmerman direct training approach[92].

This typically yields lower training cost than traditional neural networks, yet good out-of-sample performance. This is helpful in surrogate-based optimization and optimization under uncertainty, where multiple surrogates may be repeatedly constructed during the optimization process, e.g., a surrogate per response function, and a new surrogate for each optimization iteration.

The neural network is a non-parametric surface fitting method. Thus, along with Kriging (Gaussian Process) and MARS, it can be used to model data trends that have slope discontinuities as well as multiple maxima and minima. However, unlike Kriging, the neural network surrogate is not guaranteed to interpolate the data from which it was constructed.

This surrogate can be constructed from fewer than $n_{\text{quad}}$ data points, however, it is a good rule of thumb to use at least $n_{\text{quad}}$ data points when possible.
Theory

The form of the neural network model is

\[
\hat{f}(x) \approx \tanh \left\{ A_1 \tanh \left( A_0^T x + \theta_0^T \right) + \theta_1 \right\}
\]

where \( x \) is the evaluation point in \( n \)-dimensional parameter space; the terms \( A_0, \theta_0 \) are the random input layer weight matrix and bias vector, respectively; and \( A_1, \theta_1 \) are a weight vector and bias scalar, respectively, estimated from training data. These coefficients are analogous to the polynomial coefficients obtained from regression to training data. The neural network uses a cross validation-based orthogonal matching pursuit solver to determine the optimal number of nodes and to solve for the weights and offsets.

**max_nodes**

- Keywords Area
- model
- surrogate
- global
- neural_network
- max_nodes

Maximum number of hidden layer nodes

Topics

This keyword is related to the topics:

- surrogate_models

Specification

**Alias:** nodes

**Argument(s):** INTEGER

**Default:** numTrainingData - 1

Description

Limits the maximum number of hidden layer nodes in the neural network model. The default is to use one less node than the number of available training data points yielding a fully-determined linear least squares problem. However, reducing the number of nodes can help reduce overfitting and more importantly, can drastically reduce surrogate construction time when building from a large data set. (Historically, Dakota limited the number of nodes to 100.)

The keyword **max_nodes** provides an upper bound. Dakota’s orthogonal matching pursuit algorithm may further reduce the effective number of nodes in the final model to achieve better generalization to unseen points.
range

- Keywords Area
- model
- surrogate
- global
- neural_network
- range

Range for neural network random weights

Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): REAL

Description
Controls the range of the input layer random weights in the neural network model. The default range is 2.0, resulting in weights in (-1, 1). These weights are applied after the training inputs have been scaled into [-0.8, 0.8].

random_weight

- Keywords Area
- model
- surrogate
- global
- neural_network
- random_weight

(Inactive) Random weight control

Topics
This keyword is related to the topics:

- surrogate_models
6.3. MODEL

Specification

Alias: none

Argument(s): INTEGER

Description

This option is not currently in use and is likely to be removed

export_model

- Keywords Area
- model
- surrogate
- global
- neural_network
- export_model

Exports surrogate model in user-selected format

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>filename_prefix</td>
<td>formats</td>
<td>Formats for surrogate model export</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>archive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

Export the surrogate for later evaluation using the surfpack executable (bin/surfpack) or a user-developed tool. Export format is controlled using the formats specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- text_archive - Plain-text, machine-readable archive for use with the surfpack executable
CHAPTER 6. KEYWORDS AREA

- **binary_archive** - Binary, machine-readable archive for use with the surfpack executable

- **algebraic_file** - Plain-text, human-readable file intended for use with user-created tools; not compatible with the surfpack executable

- **algebraic_console** - Print the model in algebraic format to the screen; not compatible with the surfpack executable

Most global surrogates can be exported in all four formats. These include:

- Gaussian process (keyword `gaussian_process` surfpack)

- Artificial neural network (keyword `neural_network`)

- Radial basis Functions (keyword `radial_basis`)

- Polynomial (keyword `polynomial`)

However, for Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported. In addition, Dakota cannot import models that have been exported. They are strictly for use with external tools.

**Default Behavior**

No export.

**Expected Output**

Output depends on selected format; see the `formats` specification.

**Additional Discussion**

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

**filename_prefix**

- **Keywords Area**

- model

- surrogate

- global

- `neural_network`

- `export_model`

- `filename_prefix`

User-customizable portion of exported model filenames

**Topics**

This keyword is related to the topics:

- **surrogate_models**
6.3. MODEL

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

When a file-based export of a surrogate model is specified, Dakota writes one file per response, per requested format. The files are named using the pattern \{prefix\}.{response_descriptor}.{extension}.

The response descriptor portion of the pattern is filled in using the response descriptors provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format. The filename_prefix keyword is used to supply the prefix portion of the pattern.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
    filename_prefix = 'my_surrogate'
    formats
        algebraic_file
        binary_archive
```

Formats

- Keywords Area
- model
- surrogate
- global
- neural_network
- export_model
- formats

Formats for surrogate model export

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): none
<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>text_archive</td>
<td>text_archive</td>
<td>Export surrogate model to a plain-text archive file</td>
</tr>
<tr>
<td>Optional</td>
<td>binary_archive</td>
<td>binary_archive</td>
<td>Export surrogate model to a binary archive file</td>
</tr>
<tr>
<td>Optional</td>
<td>algebraic_file</td>
<td>algebraic_file</td>
<td>Export surrogate model in algebraic format to a file</td>
</tr>
<tr>
<td>Optional</td>
<td>algebraic_console</td>
<td>algebraic_console</td>
<td>Export surrogate model in algebraic format to the console</td>
</tr>
</tbody>
</table>

**Description**

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

- `text_archive`
  - Keywords Area
  - model
  - surrogate
  - global
  - neural_network
  - export_model
  - formats
  - text_archive

  Export surrogate model to a plain-text archive file

**Topics**

This keyword is related to the topics:

- surrogate_models

**Specification**

Alias: none

Argument(s): none
6.3. MODEL

Description

After the surrogate model has been built, Dakota will export it to a plain-text archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern `{prefix}.{response_descriptor}.sps`, in which ‘sps’ stands for Surfpack Surrogate. See filename_prefix for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

binary_archive

- Keywords Area
- model
- surrogate
- global
- neural_network
- export_model
- formats
- binary_archive

Export surrogate model to a binary archive file

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none
Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a binary archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern `{prefix}.{response_descriptor}.bsps`, in which ‘bsps’ stands for Binary Surfpack Surrogate. See filename_prefix for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.
algebraic_file
  - Keywords Area
  - model
  - surrogate
  - global
  - neural_network
  - export_model
  - formats
  - algebraic_file

Export surrogate model in algebraic format to a file

Topics
This keyword is related to the topics:
  - surrogate_models

Specification
Alias: none
  Argument(s): none

Description
After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern \{prefix\}.{response_descriptor}.alg. See filename_prefix for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output
The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when algebraic_console is specified.
6.3. MODEL

- **export_model**
- **formats**
- **algebraic_console**

Export surrogate model in algebraic format to the console

**Topics**

This keyword is related to the topics:

- **surrogate_models**

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the -o option) in a human-readable "algebraic" format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

**Expected Output**

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when **algebraic_file** is specified. Use of **algebraic_file** is preferred over **algebraic_console**, which exists largely to provide a measure of backward compatibility.

**radial_basis**

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **radial_basis**

Radial basis function (RBF) model

**Specification**

**Alias:** none

**Argument(s):** none
### Description

Radial basis functions $\phi$ are functions whose value typically depends on the distance from a center point, called the centroid, $c$.

The surrogate model approximation comprises a sum of $K$ weighted radial basis functions:

$$\hat{f}(x) = \sum_{k=1}^{K} w_k \phi(\|x - c_k\|)$$

These basis functions take many forms, but Gaussian kernels or splines are most common. The Dakota implementation uses a Gaussian radial basis function. The weights are determined via a linear least squares solution approach. See [67] for more details.

<table>
<thead>
<tr>
<th>Required/- Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>bases</td>
<td>bases</td>
<td>Initial number of radial basis functions</td>
</tr>
<tr>
<td>Optional</td>
<td>max_pts</td>
<td>max_pts</td>
<td>Maximum number of RBF CVT points</td>
</tr>
<tr>
<td>Optional</td>
<td>min_partition</td>
<td>min_partition</td>
<td>(Inactive) Minimum RBF partition</td>
</tr>
<tr>
<td>Optional</td>
<td>max_subsets</td>
<td>max_subsets</td>
<td>Number of trial RBF subsets</td>
</tr>
<tr>
<td>Optional</td>
<td>export_model</td>
<td>export_model</td>
<td>Exports surrogate model in user-selected format</td>
</tr>
</tbody>
</table>

Initial number of radial basis functions
6.3. MODEL

Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): INTEGER

Description
Initial number of radial basis functions. The default value is the smaller of the number of training points and 100.

max_pts

- Keywords Area
- model
- surrogate
- global
- radial_basis
- max_pts

Maximum number of RBF CVT points

Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): INTEGER

Description
Maximum number of CVT points to use in generating each RBF center. basis computing centroid of each. Defaults to 10 * (bases). Reducing this will reduce model build time.
min_partition

- Keywords Area
- model
- surrogate
- global
- radial_basis
- min_partition

(Inactive) Minimum RBF partition

Topics
This keyword is related to the topics:

- surrogate_models

Specification
Alias: none
Argument(s): INTEGER

Description
This option currently has no effect and will likely be removed.

max_subsets

- Keywords Area
- model
- surrogate
- global
- radial_basis
- max_subsets

Number of trial RBF subsets

Topics
This keyword is related to the topics:

- surrogate_models
6.3. MODEL

Specification

Alias: none

Argument(s): INTEGER

Description

Number of passes to take to identify the best subset of basis functions to use. Defaults to the smaller of \(3 \times (\text{bases})\) and 100.

\[
\text{export}_\text{model}
\]

- Keywords Area
- model
- surrogate
- global
- radial\_basis
- export\_model

Exports surrogate model in user-selected format

Topics

This keyword is related to the topics:

- surrogate\_models

Specification

Alias: none

 Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/</td>
<td></td>
<td>filename_prefix</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>formats</td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td>Formats for surrogate model export</td>
</tr>
</tbody>
</table>

Description

Export the surrogate for later evaluation using the surfpack executable (\bin\surfpack) or a user-developed tool. Export format is controlled using the \texttt{formats} specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- \texttt{text_archive} - Plain-text, machine-readable archive for use with the surfpack executable
CHAPTER 6. KEYWORDS AREA

- **binary_archive** - Binary, machine-readable archive for use with the surfpack executable
- **algebraic_file** - Plain-text, human-readable file intended for use with user-created tools; not compatible with the surfpack executable
- **algebraic_console** - Print the model in algebraic format to the screen; not compatible with the surfpack executable

Most global surrogates can be exported in all four formats. These include:

- Gaussian process (keyword `gaussian_process surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported. In addition, Dakota cannot import models that have been exported. They are strictly for use with external tools.

**Default Behavior**
No export.

**Expected Output**
Output depends on selected format; see the `formats` specification.

**Additional Discussion**
The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

**filename_prefix**

- **Keywords Area**
- model
- surrogate
- global
- radial_basis
- export_model
- filename_prefix

User-customizable portion of exported model filenames

**Topics**
This keyword is related to the topics:

- surrogate_models
6.3. MODEL

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

When a file-based export of a surrogate model is specified, Dakota writes one file per response, per requested format. The files are named using the pattern \{prefix\}.\{response_descriptor\}.\{extension\}.

The \response\_descriptor portion of the pattern is filled in using the response descriptors provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format. The \filename\_prefix keyword is used to supply the prefix portion of the pattern.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns \texttt{my_surrogate.\{response\_descriptor\}.alg} (for the algebraic files) and \texttt{my_surrogate.\{response\_descriptor\}.bsps} (for the binary files).

\begin{verbatim}
surrogate global gaussian_process surfpack export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
\end{verbatim}

Formats for surrogate model export

Topics

This keyword is related to the topics:

- surrogate\_models

Specification

Alias: none

Argument(s): none
### Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

- **text_archive**
  - Keywords Area
  - model
  - surrogate
  - global
  - radial_basis
  - export_model
  - formats
  - text_archive

  Export surrogate model to a plain-text archive file

### Topics

This keyword is related to the topics:

- surrogate_models

### Specification

**Alias:** none  
**Argument(s):** none
Description

After the surrogate model has been built, Dakota will export it to a plain-text archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern {prefix}.{response_descriptor}.sps, in which ‘sps’ stands for Surfpack Surrogate. See filename_prefix for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

binary_archive

- Keywords Area
- model
- surrogate
- global
- radial_basis
- export_model
- formats
- binary_archive

Export surrogate model to a binary archive file

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a binary archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern {prefix}.{response_descriptor}.bsps, in which ‘bsps’ stands for Binary Surfpack Surrogate. See filename_prefix for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.
Export surrogate model in algebraic format to a file

**Topics**

This keyword is related to the topics:

- surrogate_models

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern `{prefix}.{response_descriptor}.alg`. See filename_prefix for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

**Expected Output**

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when algebraic_console is specified.
6.3. MODEL

- export_model
- formats
- algebraic_console

Export surrogate model in algebraic format to the console

Topics
This keyword is related to the topics:
- surrogate_models

Specification
Alias: none
Argument(s): none

Description
After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the -o option) in a human-readable “algebraic” format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output
The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when algebraic_file is specified. Use of algebraic_file is preferred over algebraic_console, which exists largely to provide a measure of backward compatibility.

polynomial
- Keywords Area
- model
- surrogate
- global
- polynomial

Polynomial surrogate model

Specification
Alias: none
Argument(s): none
### Description

Linear, quadratic, and cubic polynomial surrogate models are available in Dakota. The utility of the simple polynomial models stems from two sources:

- over a small portion of the parameter space, a low-order polynomial model is often an accurate approximation to the true data trends
- the least-squares procedure provides a surface fit that smooths out noise in the data.

Local surrogate-based optimization methods (surrogate\_based\_local) are often successful when using polynomial models, particularly quadratic models. However, a polynomial surface fit may not be the best choice for modeling data trends globally over the entire parameter space, unless it is known a priori that the true data trends are close to linear, quadratic, or cubic. See[63] for more information on polynomial models.

### Theory

The form of the linear polynomial model is

$$ \hat{f}(\mathbf{x}) \approx c_0 + \sum_{i=1}^{n} c_i x_i $$

the form of the quadratic polynomial model is:

$$ \hat{f}(\mathbf{x}) \approx c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j \geq i}^{n} c_{ij} x_i x_j $$

and the form of the cubic polynomial model is:

$$ \hat{f}(\mathbf{x}) \approx c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j \geq i}^{n} c_{ij} x_i x_j + \sum_{i=1}^{n} \sum_{j \geq i}^{n} \sum_{k \geq j}^{n} c_{ijk} x_i x_j x_k $$

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>polynomial order (Group 1)</td>
<td>basis_order</td>
<td>Polynomial order</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td></td>
<td>linear</td>
<td>Use a linear polynomial or trend function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>quadratic</td>
<td>Use a quadratic polynomial or trend function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cubic</td>
<td>Use a cubic polynomial</td>
</tr>
<tr>
<td>Optional</td>
<td>export_model</td>
<td></td>
<td>Exports surrogate model in user-selected format</td>
</tr>
</tbody>
</table>
In all of the polynomial models, \( \hat{f}(x) \) is the response of the polynomial model; the \( x_i, x_j, x_k \) terms are the components of the \( n \)-dimensional design parameter values; the \( c_0, c_i, c_{ij}, c_{ijk} \) terms are the polynomial coefficients, and \( n \) is the number of design parameters. The number of coefficients, \( n_c \), depends on the order of polynomial model and the number of design parameters. For the linear polynomial:

\[
    n_{c\text{linear}} = n + 1
\]

for the quadratic polynomial:

\[
    n_{c\text{quad}} = \frac{(n + 1)(n + 2)}{2}
\]

and for the cubic polynomial:

\[
    n_{c\text{cubic}} = \frac{(n^3 + 6n^2 + 11n + 6)}{6}
\]

There must be at least \( n_c \) data samples in order to form a fully determined linear system and solve for the polynomial coefficients. In Dakota, a least-squares approach involving a singular value decomposition numerical method is applied to solve the linear system.

**basis_order**

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **polynomial**
- **basis_order**

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

The polynomial order for the polynomial regression model (default = 2).
• polynomial
• linear

Use a linear polynomial or trend function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent page

• quadratic
  • Keywords Area
  • model
  • surrogate
  • global
  • polynomial
  • quadratic

Use a quadratic polynomial or trend function

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See parent page

• cubic
  • Keywords Area
  • model
  • surrogate
  • global
  • polynomial
  • cubic

Use a cubic polynomial
6.3. MODEL

Specification

Alias: none
Argument(s): none

Description

See parent page

export_model

- Keywords Area
- model
- surrogate
- global
- polynomial
- export_model

Exports surrogate model in user-selected format

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>filename_prefix</td>
<td>User-customizable portion of exported model filenames</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>formats</td>
<td>Formats for surrogate model export</td>
</tr>
</tbody>
</table>

Description

Export the surrogate for later evaluation using the surfpack executable (bin/surfpack) or a user-developed tool. Export format is controlled using the formats specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- text_archive - Plain-text, machine-readable archive for use with the surfpack executable
• **binary_archive** - Binary, machine-readable archive for use with the surfpack executable

• **algebraic_file** - Plain-text, human-readable file intended for use with user-created tools; not compatible with the surfpack executable

• **algebraic_console** - Print the model in algebraic format to the screen; not compatible with the surfpack executable

Most global surrogates can be exported in all four formats. These include:

• Gaussian process (keyword `gaussian_process` surfpack)

• Artificial neural network (keyword `neural_network`)

• Radial basis Functions (keyword `radial_basis`)

• Polynomial (keyword `polynomial`)

However, for Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported. In addition, Dakota cannot import models that have been exported. They are strictly for use with external tools.

**Default Behavior**
No export.

**Expected Output**
Output depends on selected format; see the formats specification.

**Additional Discussion**
The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

**filename_prefix**

• **Keywords Area**

• **model**

• **surrogate**

• **global**

• **polynomial**

• **export_model**

• **filename_prefix**

User-customizable portion of exported model filenames

**Topics**
This keyword is related to the topics:

• **surrogate_models**
6.3. MODEL

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

When a file-based export of a surrogate model is specified, Dakota writes one file per response, per requested format. The files are named using the pattern {prefix}.{response_descriptor}.{extension}.

The response_descriptor portion of the pattern is filled in using the response descriptors provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format. The filename_prefix keyword is used to supply the prefix portion of the pattern.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns my_surrogate.{response_descriptor}.alg (for the algebraic files) and my_surrogate.{response_descriptor}.bsps (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive

  formats
    • Keywords Area
    • model
    • surrogate
    • global
    • polynomial
    • export_model
    • formats
  Formats for surrogate model export
```

Topics

This keyword is related to the topics:

• surrogate_models

Specification

Alias: none

Argument(s): none
### Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

- **text_archive**
  - Keywords Area
  - model
  - surrogate
  - global
  - polynomial
  - export_model
  - formats
  - text_archive

  Export surrogate model to a plain-text archive file

### Topics

This keyword is related to the topics:

- surrogate_models

### Specification

**Alias**: none
**Argument(s)**: none
6.3. **MODEL**

**Description**

After the surrogate model has been built, Dakota will export it to a plain-text archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack Surrogate. See `filename_prefix` for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.

- **binary_archive**
  - **Keywords Area**
  - **model**
  - **surrogate**
  - **global**
  - **polynomial**
  - **export_model**
  - **formats**
  - **binary_archive**

Export surrogate model to a binary archive file

**Topics**

This keyword is related to the topics:

- **surrogate_models**

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

After the surrogate model has been built, Dakota will export it to a binary archival file suitable only for use with the surfpack executable (bin/surfpack). The file is named using the pattern `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for Binary Surfpack Surrogate. See `filename_prefix` for further information about exported surrogate file naming.

The Dakota examples folder includes a demonstration of using the surfpack executable with an exported model file.
**algebraic_file**

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **polynomial**
- **export_model**
- **formats**
- **algebraic_file**

Export surrogate model in algebraic format to a file

**Topics**

This keyword is related to the topics:

- **surrogate_models**

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern `{prefix}.{response_descriptor}.alg`. See `filename_prefix` for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

**Expected Output**

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when `algebraic_console` is specified.
6.3. MODEL

- export_model
- formats
- algebraic_console

Export surrogate model in algebraic format to the console

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the -o option) in a human-readable "algebraic" format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when algebraic_file is specified. Use of algebraic_file is preferred over algebraic_console, which exists largely to provide a measure of backward compatibility.

domain_decomposition

- Keywords Area
- model
- surrogate
- global
- domain_decomposition

Piecewise Domain Decomposition for Global Surrogate Models

Specification

Alias: none

Argument(s): none
### Description

Typical regression techniques use all available sample points to build continuous approximations of the underlying function.

An alternative option is to use piecewise decomposition to locally approximate the function at some point using a few sample points from its neighborhood only. This option currently supports Polynomial Regression, Gaussian Process (GP) Interpolation, and Radial Basis Functions (RBF) Regression. It requires a decomposition cell type (currently set to be Voronoi cells). Optional parameters are: the number of layers of neighbors used to solve the regression problem (default is one layer), and an optional discontinuity detection capability (identified by a user-input jump or gradient threshold).

The method can also make use of the gradient and Hessian information, if available. The user needs to specify the keyword `user_derivatives`.

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>cell_type</td>
<td>Type of the Geometric Cells Used for the Piecewise Decomposition Option of Global Surrogates</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>support_layers</td>
<td>Optional Number of Support Layers for the Piecewise Decomposition Option of Global Surrogates</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>discontinuity_detection</td>
<td>Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates</td>
</tr>
</tbody>
</table>

- **cell_type**
  - Keywords Area
  - model
  - surrogate
  - global
  - domain_decomposition
6.3. MODEL

- **cell_type**
  
  Type of the Geometric Cells Used for the Piecewise Decomposition Option of Global Surrogates

**Specification**

**Alias:** none

**Argument(s):** STRING

**Description**

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

This option requires a decomposition cell type that can vary from structured grid boxes, to polygonal Voronoi cells. Currently, this option only supports Voronoi cells.

**support_layers**

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **domain_decomposition**
- **support_layers**

Optional Number of Support Layers for the Piecewise Decomposition Option of Global Surrogates

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Description**

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The neighborhood of a cell is parameterized via a number of (support layers). The default value is set to one layer of neighbors (cells that share direct edges with the cell under study). One more support layer would include the neighbors of that cells neighbors, and so on.

**discontinuity_detection**

- **Keywords Area**
- **model**
- **surrogate**
- **global**
Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

Specification

Alias: none  
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump</td>
<td>Group 1</td>
<td>jump_threshold</td>
<td>Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates</td>
</tr>
<tr>
<td>gradient_threshold</td>
<td></td>
<td></td>
<td>Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates</td>
</tr>
</tbody>
</table>

Description

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The domain decomposition algorithm supports an optional discontinuity detection capability where seeds across a user-input discontinuity threshold are not considered neighbors when building the approximate connectivity Delaunay graph. Alternatively, the domain is split into patches that trap discontinuities between them. This capability is specified by either jump or gradient threshold values in the input spec.

- jump_threshold

  - Keywords Area
  - model
  - surrogate
Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

**Specification**

**Alias:** none

**Argument(s):** REAL

**Description**

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The domain decomposition algorithm supports an optional discontinuity detection capability where seeds across a user-input discontinuity threshold are not considered neighbors when building the approximate connectivity Delaunay graph. Alternatively, the domain is split into patches that trap discontinuities between them. This capability can be specified using a jump threshold value in the input spec.

**gradient_threshold**

**Keywords Area**

**model**

**surrogate**

**global**

**domain_decomposition**

**discontinuity_detection**

**gradient_threshold**

Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

**Specification**

**Alias:** none

**Argument(s):** REAL
CHAPTER 6. KEYWORDS AREA

Description
The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The domain decomposition algorithm supports an optional discontinuity detection capability where seeds across a user-input discontinuity threshold are not considered neighbors when building the approximate connectivity Delaunay graph. Alternatively, the domain is split into patches that trap discontinuities between them. This capability can be specified using a gradient threshold value in the input spec.

total_points
- Keywords Area
- model
- surrogate
- global
- total_points
  Specified number of training points

Specification
Alias: none
Argument(s): INTEGER
Default: recommended_points

Description
See parent page.

minimum_points
- Keywords Area
- model
- surrogate
- global
- minimum_points
  Construct surrogate with minimum number of points

Specification
Alias: none
Argument(s): none

Description
The minimum is d+1, for d input dimensions, except for polynomials. See parent page.
6.3. MODEL

recommended_points

- Keywords Area
- model
- surrogate
- global
- recommended_points

Construct surrogate with recommended number of points

Specification

Alias: none
Argument(s): none

Description

This is the default option. It requires $5d$ build points for $d$ input dimensions, except for polynomial models. See parent page.

dace_method_pointer

- Keywords Area
- model
- surrogate
- global
- dace_method_pointer

Specify a method to gather training data

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none
Argument(s): STRING
Default: no design of experiments data
Description

The number of training points and the sources are specified on global, as well as the number of new training points required.

New training points are gathered by running the "truth" model using the method specified by dace_method_pointer. The DACE method will only be invoked if it has new samples to perform, and if new samples are required and no DACE iterator has been provided, an error will result.

The dace_method_pointer points to design of experiments method block used to generate truth model data.

Permissible methods include: Monte Carlo (random) sampling, Latin hypercube sampling, orthogonal array sampling, central composite design sampling, and Box-Behnken sampling.

Note that the number of samples specified in the method block may be overwritten, if the requested number of samples is less than minimum_points.

actual_model_pointer

• Keywords Area
• model
• surrogate
• global
• actual_model_pointer

A surrogate model pointer that guides a method to whether it should use a surrogate model or compute truth function evaluations

Specification

Alias: none

Argument(s): STRING

Description

Dakota methods use global surrogate models to compute surrogate function approximations. They also need to know the true function evaluations. A global surrogate model now must have an actual_model_pointer keyword to decide for the method whether to evaluate the global surrogate model, or compute the true function evaluations if actual_model_pointer = TRUTH.

reuse_points

• Keywords Area
• model
• surrogate
• global
• reuse_points

Surrogate model training data reuse control
6.3. MODEL

Topics

This keyword is related to the topics:

- surrogate_models

Specification

Alias: reuse_samples

Argument(s): none

Default: all for import; none otherwise

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>all</td>
<td>Option for reuse_points</td>
</tr>
<tr>
<td>Required(Choose One)</td>
<td></td>
<td>region</td>
<td>Option for reuse_points</td>
</tr>
<tr>
<td></td>
<td></td>
<td>none</td>
<td>Option for reuse_points</td>
</tr>
</tbody>
</table>

Description

Dakota’s global surrogate methods rely on training data, which can either come from evaluation of a “truth” model, which is generated by the method specified with dace_method_pointer, from a file of existing training data, identified by import_build_points_file, or both.

The reuse_points keyword controls the amount of training data used in building a surrogate model, either initially, or during iterative rebuild, as in surrogate-based optimization. If import_build_points_file is specified, reuse_points controls how the file contents are used. If used during iterative rebuild, it controls what data from previous surrogate builds is reused in building the current model.

- all (default for file import) - use all points in the file or available from previous builds
- region - use only the points falling in the current trust region (see surrogate_based_local)
- none (default when no import) - ignore the contents of the file or previous build points, and gather new training data using the specified DACE method

all

Keywords Area

- model
- surrogate
- global
- reuse_points
- all

Option for reuse_points
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
  Argument(s): none

Description
This is described on the parent page.

region
  • Keywords Area
  • model
  • surrogate
  • global
  • reuse_points
  • region
Option for reuse_points

Specification
Alias: none
  Argument(s): none

Description
This is described on the parent page.

none
  • Keywords Area
  • model
  • surrogate
  • global
  • reuse_points
  • none
Option for reuse_points

Specification
Alias: none
  Argument(s): none
6.3. MODEL

Description
This is described on the parent page.

**import_build_points_file**

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **import_build_points_file**

File containing points you wish to use to build a surrogate

**Specification**

**Alias:** import_points_file samples_file

**Argument(s):** STRING

**Default:** no point import from a file

<table>
<thead>
<tr>
<th>Required/-.Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional/Choose One</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

**Description**

The **import_build_points_file** allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

**Default Behavior**

Be default, methods do not import points from a file.

**Usage Tips**

Dakota parses input files without regard to whitespace, but the **import_build_points_file** must be in one of three formats:
**annotated** (default)

**custom_annotated**

**freeform**

### Examples

```plaintext
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'

annotated

- Keywords Area
- model
- surrogate
- global
- import_build_points_file
- annotated

Selects annotated tabular file format

### Topics

This keyword is related to the topics:

- **file_formats**

### Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated

### Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

**Default Behavior**

By default, Dakota imports and exports tabular files in annotated format. The **annotated** keyword can be used to explicitly specify this.

**Usage Tips**

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify **custom_annotated** header eval_id
Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

**Examples**

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated

Resulting tabular file:

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>interface</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO_ID</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>NO_ID</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.000196404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>NO_ID</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

custom_annotated

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **import_build_points_file**
- **custom_annotated**

Selects custom-annotated tabular file format

**Topics**

This keyword is related to the topics:

- **file_formats**

**Specification**

**Alias:** none  
**Argument(s):** none
## Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id   x1     x2    obj_fn nln_ineq_con_1 nln_ineq_con_2
1 0.9 1.1 0.0002 0.26 0.76
2 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...```

### Table

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
header

- Keywords Area
- model
- surrogate
- global
- import_build_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See description of parent `custom_annotated`

**eval_id**

- Keywords Area
- model
- surrogate
- global
- import_build_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See description of parent `custom_annotated`
interface_id

- Keywords Area
- model
- surrogate
- global
- import_build_points_file
- custom_annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See description of parent `custom_annotated`

**freeform**

- Keywords Area
- model
- surrogate
- global
- import_build_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** annotated
6.3. MODEL

Description
A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior
The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips
- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9  1.1  0.0002  0.26  0.76  
0.90009  1.1  0.0001996404857  0.2601620081  0.759955  
0.89991  1.1  0.0002003604863  0.2598380081  0.760045  
...  
```

active_only

- Keywords Area
- model
- surrogate
- global
- import_build_points_file
- active_only

Import only active variables from tabular data file

Topics
This keyword is related to the topics:

- file_formats
**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

**export_approx_points_file**

- **Keywords Area**
- **model**
- **surrogate**
- **global**
- **export_approx_points_file**

Output file for evaluations of a surrogate model

**Specification**

**Alias:** export_points_file  
**Argument(s):** STRING  
**Default:** no point export to a file

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform file format</td>
</tr>
</tbody>
</table>

**Description**

The `export_approx_points_file` keyword allows the user to specify a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate’s predicted approximation to the truth model responses at those points.
Usage Tips
Dakota exports tabular data in one of three formats:

- annotated (default)
- custom_annotated
- freeform

annotated
- Keywords Area
- model
- surrogate
- global
- export_approx_points_file
- annotated

Selects annotated tabular file format

Topics
This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated

Description
An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior
By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.
Examples
Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```environment
tabular_data
  tabular_data_file = 'dakota_summary.dat'
annotated```

Resulting tabular file:

```
%eval_id interface x1 x2  obj_fn nln_ineq_con_1 nln_ineq_con_2
1    NO_ID    0.9  1.1 0.0002 0.26 0.76
2    NO_ID  0.90009  1.1 0.0001996404857 0.2601620081 0.759955
3    NO_ID  0.89991  1.1 0.0002003604863 0.2598380081 0.760045
...```

custom_annotated

- Keywords Area
- model
- surrogate
- global
- export_approx_points_file
- custom_annotated

Selects custom-annotated tabular file format

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th></th>
<th>Description</th>
</tr>
</thead>
</table>
Optional | header | | Enable header row in custom-annotated tabular file |
6.3. MODEL

<table>
<thead>
<tr>
<th>Optional</th>
<th>eval_id</th>
<th>Enable evaluation ID column in custom-annotated tabular file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

### Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

#### Default Behavior

The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom.annotated, followed by options, in the relevant export/import context.

#### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

#### Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom.annotated header eval_id

Resulting tabular file:

<table>
<thead>
<tr>
<th>eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj fn</th>
<th>nln.ineq_con_1</th>
<th>nln.ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
</tbody>
</table>
...```

header

- Keywords Area
- model
- surrogate
• global

• export_approx_points_file

• custom.annotated

• header

Enable header row in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See description of parent `custom.annotated`

**eval_id**

• Keywords Area

• model

• surrogate

• global

• export_approx_points_file

• custom.annotated

• eval_id

Enable evaluation ID column in custom-annotated tabular file

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

See description of parent `custom.annotated`
6.3. MODEL

interface_id

- Keywords Area
- model
- surrogate
- global
- export_approx_points_file
- custom.annotated
- interface_id

Enable interface ID column in custom-annotated tabular file

**Specification**

**Alias:** none
**Argument(s):** none

**Description**

See description of parent custom.annotated

**freeform**

- Keywords Area
- model
- surrogate
- global
- export_approx_points_file
- freeform

Selects freeform file format

**Topics**

This keyword is related to the topics:

- file.formats

**Specification**

**Alias:** none
**Argument(s):** none
**Default:** annotated
Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
freeform
```

Resulting tabular file:

```
0.9  1.1  0.0002  0.26  0.76
0.90009  1.1  0.0001996404857  0.2601620081  0.759955
0.89991  1.1  0.0002003604863  0.2598380081  0.760045
...```

use_derivatives

- Keywords Area
- model
- surrogate
- global
- use_derivatives

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only
6.3. Model

Description

The use_derivatives flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

correction

- Keywords Area
- model
- surrogate
- global
- correction

Correction approaches for surrogate models

Specification

Alias: none
Argument(s): none
Default: no surrogate correction

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>correction order (Group 1)</td>
<td>zeroth_order</td>
<td>Specify that truth values must be matched.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>first_order</td>
<td>Specify that truth values and gradients must be matched.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>second_order</td>
<td>Specify that truth values, gradients and Hessians must be matched.</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>correction type (Group 2)</td>
<td>additive</td>
<td>Additive correction factor for local surrogate accuracy</td>
</tr>
</tbody>
</table>
### Description

Some of the surrogate model types support the use of correction factors that improve the local accuracy of the surrogate models.

The correction specification specifies that the approximation will be corrected to match truth data, either matching truth values in the case of zeroth-order matching, matching truth values and gradients in the case of first-order matching, or matching truth values, gradients, and Hessians in the case of second-order matching. For additive and multiplicative corrections, the correction is local in that the truth data is matched at a single point, typically the center of the approximation region. The additive correction adds a scalar offset (zeroth-order), a linear function (first-order), or a quadratic function (second-order) to the approximation to match the truth data at the point, and the multiplicative correction multiplies the approximation by a scalar (zeroth-order), a linear function (first-order), or a quadratic function (second-order) to match the truth data at the point. The additive first-order case is due to[57] and the multiplicative first-order case is commonly known as beta correction[40]. For the combined correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The combined correction is then a multipoint correction, as opposed to the local additive and multiplicative corrections. Each of these correction capabilities is described in detail in[24].

The correction factors force the surrogate models to match the true function values and possibly true function derivatives at the center point of each trust region. Currently, Dakota supports either zeroth-, first-, or second-order accurate correction methods, each of which can be applied using either an additive, multiplicative, or combined correction function. For each of these correction approaches, the correction is applied to the surrogate model and the corrected model is then interfaced with whatever algorithm is being employed. The default behavior is that no correction factor is applied.

The simplest correction approaches are those that enforce consistency in function values between the surrogate and original models at a single point in parameter space through use of a simple scalar offset or scaling applied to the surrogate model. First-order corrections such as the first-order multiplicative correction (also known as beta correction[15]) and the first-order additive correction[57] also enforce consistency in the gradients and provide a much more substantial correction capability that is sufficient for ensuring provable convergence in SBO algorithms. SBO convergence rates can be further accelerated through the use of second-order corrections which also enforce consistency in the Hessians[24], where the second-order information may involve analytic, finite-difference, or quasi-Newton Hessians.

Correcting surrogate models with additive corrections involves

$$ f_{hi,a}(\mathbf{x}) = f_{lo}(\mathbf{x}) + \alpha(\mathbf{x}) \quad (6.1) $$

where multifidelity notation has been adopted for clarity. For multiplicative approaches, corrections take the form

$$ f_{hi,o}(\mathbf{x}) = f_{lo}(\mathbf{x})/\beta(\mathbf{x}) \quad (6.2) $$

<table>
<thead>
<tr>
<th>Correction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplicative</td>
<td>Multiplicative correction factor for local surrogate accuracy.</td>
</tr>
<tr>
<td>combined</td>
<td>Multipoint correction for a hierarchical surrogate.</td>
</tr>
</tbody>
</table>
where, for local corrections, $\alpha(x)$ and $\beta(x)$ are first or second-order Taylor series approximations to the exact correction functions:

\[
\alpha(x) = A(x_c) + \nabla A(x_c)^T (x - x_c) + \frac{1}{2}(x - x_c)^T \nabla^2 A(x_c) (x - x_c) \tag{6.3}
\]
\[
\beta(x) = B(x_c) + \nabla B(x_c)^T (x - x_c) + \frac{1}{2}(x - x_c)^T \nabla^2 B(x_c) (x - x_c) \tag{6.4}
\]

where the exact correction functions are

\[
A(x) = f_{hi}(x) - f_{lo}(x) \tag{6.5}
\]
\[
B(x) = \frac{f_{hi}(x)}{f_{lo}(x)} \tag{6.6}
\]

Refer to[24] for additional details on the derivations.

A combination of additive and multiplicative corrections can provide for additional flexibility in minimizing the impact of the correction away from the trust region center. In other words, both additive and multiplicative corrections can satisfy local consistency, but through the combination, global accuracy can be addressed as well. This involves a convex combination of the additive and multiplicative corrections:

\[
\hat{f}_{hi}(x) = \gamma \hat{f}_{hi}(x) + (1 - \gamma) \hat{f}_{hi}(x)
\]

where $\gamma$ is calculated to satisfy an additional matching condition, such as matching values at the previous design iterate.

It should be noted that in both first order correction methods, the function $\hat{f}(x)$ matches the function value and gradients of $f_t(x)$ at $x = x_c$. This property is necessary in proving that the first order-corrected SBO algorithms are provably convergent to a local minimum of $f_t(x)$. However, the first order correction methods are significantly more expensive than the zeroth order correction methods, since the first order methods require computing both $\nabla f_t(x_c)$ and $\nabla f_s(x_c)$. When the SBO strategy is used with either of the zeroth order correction methods, or with no correction method, convergence is not guaranteed to a local minimum of $f_t(x)$. That is, the SBO strategy becomes a heuristic optimization algorithm. From a mathematical point of view this is undesirable, but as a practical matter, the heuristic variants of SBO are often effective in finding local minima.

Usage guidelines

- Both the additive zeroth order and multiplicative zeroth order correction methods are "free" since they use values of $f_t(x_c)$ that are normally computed by the SBO strategy.

- The use of either the additive first order method or the multiplicative first order method does not necessarily improve the rate of convergence of the SBO algorithm.

- When using the first order correction methods, the gradient-related response keywords must be modified to allow either analytic or numerical gradients to be computed. This provides the gradient data needed to compute the correction function.

- For many computationally expensive engineering optimization problems, gradients often are too expensive to obtain or are discontinuous (or may not exist at all). In such cases the heuristic SBO algorithm has been an effective approach at identifying optimal designs[35].
zeroth_order

- Keywords Area
- model
- surrogate
- global
- correction
- zeroth_order

Specify that truth values must be matched.

**Specification**

Alias: none

Argument(s): none

**Description**

The correction specification specifies that the approximation will be corrected to match truth data. The keyword *zeroth_order* matching ensures that truth values are matched.

first_order

- Keywords Area
- model
- surrogate
- global
- correction
- first_order

Specify that truth values and gradients must be matched.

**Specification**

Alias: none

Argument(s): none

**Description**

This correction specification specifies that the approximation will be corrected to match truth data. The keyword *first_order* matching ensures that truth values and gradients are matched.
second_order

- Keywords Area
- model
- surrogate
- global
- correction
- second_order

Specify that truth values, gradients and Hessians must be matched.

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The correction specification specifies that the approximation will be corrected to match truth data. The keyword second_order matching ensures that truth values, gradients and Hessians are matched.

additive

- Keywords Area
- model
- surrogate
- global
- correction
- additive

Additive correction factor for local surrogate accuracy

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Use an additive correction factor to improve the local accuracy of a surrogate.
CHAPTER 6. KEYWORDS AREA

**multiplicative**

- Keywords Area
- model
- surrogate
- global
- correction
- multiplicative

Multiplicative correction factor for local surrogate accuracy.

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

Use a multiplicative correction factor to improve the local accuracy of a surrogate.

**combined**

- Keywords Area
- model
- surrogate
- global
- correction
- combined

Multipoint correction for a hierarchical surrogate

**Specification**

**Alias:** none  
**Argument(s):** none

**Description**

For the combined correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The combined correction is then a multipoint correction, as opposed to the local additive and multiplicative corrections.
6.3. **MODEL**

**metrics**
- Keywords Area
- model
- surrogate
- global
- metrics

Compute surrogate quality metrics

**Topics**
This keyword is related to the topics:
- surrogate_models

**Specification**

**Alias:** diagnostics  
**Argument(s):** STRINGLIST  
**Default:** No diagnostics

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>cross_validation</td>
<td>press</td>
</tr>
</tbody>
</table>

**Description**

A variety of diagnostic metrics are available to assess the goodness of fit of a global surrogate to its training data. The default diagnostics are:

- root_mean_squared
- mean_abs
- rsquared

Additional available diagnostics include

- sum_squared
- mean_squared
- sum_abs
- max_abs

The keywords **press** and **cross_validation** further specify leave-one-out or k-fold cross validation, respectively, for all of the active metrics from above.
Theory

Most of these diagnostics refer to some operation on the residuals (the difference between the surrogate model and the truth model at the data points upon which the surrogate is built).

For example, \texttt{sum\_squared} refers to the sum of the squared residuals, and \texttt{mean\_abs} refers to the mean of the absolute value of the residuals. \texttt{rsquared} refers to the R-squared value typically used in regression analysis (the proportion of the variability in the response that can be accounted for by the surrogate model). Care should be taken when interpreting metrics, for example, errors may be near zero for interpolatory models or \texttt{rsquared} may not be applicable for non-polynomial models.

\begin{verbatim}
cross\_validation
  * Keywords Area
  * model
  * surrogate
  * global
  * metrics
  * cross\_validation

Perform k-fold cross validation
\end{verbatim}

Topics

This keyword is related to the topics:

\begin{verbatim}
  * surrogate\_models
\end{verbatim}

Specification

\textbf{Alias:} none

\textbf{Argument(s):} none

\textbf{Default:} No cross validation

\begin{verbatim}
<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional Optional\texttt{(Choose One)}</td>
<td>Group 1</td>
<td>folds</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>percent</td>
<td>Number of cross validation folds</td>
</tr>
</tbody>
</table>

Percent data per cross validation fold
\end{verbatim}

Description

General k-fold cross validation may be performed by specifying \texttt{cross\_validation}. The cross-validation statistics will be calculated for all metrics.

Cross validation may further specify:

\begin{verbatim}
  * \texttt{folds}, the number of folds into which to divide the build data (between 2 and number of data points) or
\end{verbatim}
6.3. **MODEL**

- **percent**, the fraction of data (between 0 and 0.5) to use in each fold.

These will be adjusted as needed based on the number of available training points. The default number of folds \( k = 10 \), or 0.1

```
folds
  * Keywords Area
  * model
  * surrogate
  * global
  * metrics
  * cross_validation
  * folds
```

**Specification**

**Alias:** none

- **Argument(s):** INTEGER
- **Default:** 10

**Description**

Number of folds (partitions) of the training data to use in cross validation (default 10).

```
percent
  * Keywords Area
  * model
  * surrogate
  * global
  * metrics
  * cross_validation
  * percent
```

Percent data per cross validation fold

**Specification**

**Alias:** none

- **Argument(s):** REAL
- **Default:** 0.1
CHAPTER 6. KEYWORDS AREA

Description
Percent of the training data to use in each cross validation fold (default 0.1).

press
  • Keywords Area
  • model
  • surrogate
  • global
  • metrics
  • press

Leave-one-out cross validation

Specification
Alias: none
Argument(s): none
Default: No PRESS cross validation

Description
Leave-one-out (PRESS) cross validation may be performed by specifying press. The cross-validation statistics will be calculated for all metrics.

import_challenge_points_file
  • Keywords Area
  • model
  • surrogate
  • global
  • import_challenge_points_file

Datafile of points to assess surrogate quality

Specification
Alias: challenge_points_file
Argument(s): STRING
Default: no user challenge data
### Required/Optional

**Option** (Choose One)

- **Required/Optional**
- **Description of Group**
- **Table format**
- **Dakota Keyword**
- **Dakota Keyword Description**

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>annotated</td>
<td>Selects annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>freeform</td>
<td>Selects freeform tabular file format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>active_only</td>
<td>Import only active variables from tabular data file</td>
</tr>
</tbody>
</table>

### Description

Specifies a data file containing variable and response (truth) values, in one of three formats:

- annotated (default)
- custom_annotated
- freeform

The surrogate is evaluated at the points in the file, and the surrogate (approximate) responses are compared against the truth results from the file. All metrics specified with `metrics` will be computed for the challenge data.

- annotated
  - Keywords Area
  - model
  - surrogate
  - global
  - import_challenge_points_file
  - annotated
    Selects annotated tabular file format

### Topics

This keyword is related to the topics:

- file_formats
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include interface_id, specify custom_annotated header eval_id
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```bash
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
annotated
```

Resulting tabular file:

```
%eval_id interface x1 x2 obj_fn nln_ineq_con_1 nln_ineq_con_2
1 NO_ID 0.9 1.1 0.0002 0.26 0.76
2 NO_ID 0.90009 1.1 0.0001996404857 0.2601620081 0.759955
3 NO_ID 0.89991 1.1 0.0002003604863 0.2598380081 0.760045
...
```

• Keywords Area
• model
• surrogate
• global
• import_challenge_points_file
• custom_annotated

Selects custom-annotated tabular file format
6.3. MODEL

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>header</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>eval_id</td>
<td>Enable evaluation ID column in custom-annotated tabular file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>interface_id</td>
<td>Enable interface ID column in custom-annotated tabular file</td>
</tr>
</tbody>
</table>

Description
A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, eval_id column, and interface_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior
The annotated format is the default for tabular export/import. To control which header row and columns are in the input/output, specify custom_annotated, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples
Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:
environment
tabular_data
tabular_data_file = 'dakota_summary.dat'
custom_annotated header eval_id

Resulting tabular file:

<table>
<thead>
<tr>
<th>%eval_id</th>
<th>x1</th>
<th>x2</th>
<th>obj_fn</th>
<th>nln_ineq_con_1</th>
<th>nln_ineq_con_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
<td>1.1</td>
<td>0.0002</td>
<td>0.26</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>0.90009</td>
<td>1.1</td>
<td>0.0001996404857</td>
<td>0.2601620081</td>
<td>0.759955</td>
</tr>
<tr>
<td>3</td>
<td>0.89991</td>
<td>1.1</td>
<td>0.0002003604863</td>
<td>0.2598380081</td>
<td>0.760045</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

header

- Keywords Area
- model
- surrogate
- global
- import_challenge_points_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

Specification

Alias: none
Argument(s): none

Description

See description of parent custom_annotated

eval_id

- Keywords Area
- model
- surrogate
- global
- import_challenge_points_file
- custom_annotated
- eval_id

Enable evaluation ID column in custom-annotated tabular file
6.3. MODEL

**Specification**

Alias: none  
Argument(s): none

**Description**

See description of parent `custom_annotated`

```plaintext
interface_id
  • Keywords Area
  • model
  • surrogate
  • global
  • import_challenge_points_file
  • custom_annotated
  • interface_id

Enable interface ID column in custom-annotated tabular file
```

**Specification**

Alias: none  
Argument(s): none

**Description**

See description of parent `custom_annotated`

```plaintext
freeform
  • Keywords Area
  • model
  • surrogate
  • global
  • import_challenge_points_file
  • freeform

Selects freeform file format
```

**Topics**

This keyword is related to the topics:

```plaintext
• file_formats
```
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

    Argument(s): none
    Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The annotated format is the default for tabular export/import. To change this behavior, specify freeform in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In freeform, the num_rows x num_cols total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the freeform option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```plaintext
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
    0.9  1.1  0.0002  0.26  0.76
  0.90009  1.1  0.0001996404857  0.2601620081  0.759955
  0.89991  1.1  0.0002003604863  0.2598380081  0.760045
  ...
```

active_only

- Keywords Area
- model
- surrogate
- global
- import_challenge_points_file
- active_only

Import only active variables from tabular data file
6.3. MODEL

Topics
This keyword is related to the topics:

- file_formats

Specification
Alias: none
Argument(s): none

Description
By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

multipoint

- Keywords Area
- model
- surrogate
- multipoint

Construct a surrogate from multiple existing training points

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>tana</td>
<td>Local multi-point model via two-point nonlinear approximation</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>actual_model_pointer</td>
<td>Pointer to specify a &quot;truth&quot; model, from which to construct a surrogate</td>
</tr>
</tbody>
</table>
Description

Multipoint approximations use data from previous design points to improve the accuracy of local approximations. The data often comes from the current and previous iterates of a minimization algorithm.

Currently, only the Two-point Adaptive Nonlinearity Approximation (TANA-3) method of [91] is supported with the \texttt{tana} keyword.

The truth model to be used to generate the value/gradient data used in the approximation is identified through the required \texttt{actual\_model\_pointer} specification.

See Also

These keywords may also be of interest:

- \texttt{local}
- \texttt{global}
- \texttt{hierarchical}

\texttt{tana}

- Keywords Area
- \texttt{model}
- \texttt{surrogate}
- \texttt{multipoint}
- \texttt{tana}

Local multi-point model via two-point nonlinear approximation

Specification

\texttt{Alias: none}

\texttt{Argument(s): none}

Description

TANA stands for Two Point Adaptive Nonlinearity Approximation.

The TANA-3 method[91] is a multipoint approximation method based on the two point exponential approximation[25]. This approach involves a Taylor series approximation in intermediate variables where the powers used for the intermediate variables are selected to match information at the current and previous expansion points.

Theory

The form of the TANA model is:

\[
\hat{f}(x) \approx f(x_2) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x_2) \frac{x_i^{1-p_i}}{p_i} (x_i^p_i - x^p_{i,2}) + \frac{1}{2} \epsilon(x) \sum_{i=1}^{n} (x_i^p_i - x^p_{i,2})^2
\]

where \( n \) is the number of variables and:
\[ p_i = 1 + \ln \left[ \frac{\frac{\partial f}{\partial x_i}(x_1)}{\frac{\partial f}{\partial x_i}(x_2)} \right] \ln \left[ \frac{x_{1,i}}{x_{2,i}} \right] \] 

\[ \epsilon(x) = \frac{H}{\sum_{i=1}^{n} (x_{i,1} - x_{i,1})^2 + \sum_{i=1}^{n} (x_{i,2} - x_{i,2})^2} H = 2 \left[ f(x_1) - f(x_2) - \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x_2) \frac{x_{1,i} - x_{2,i}}{p_i} \right] \]

and \( x_2 \) and \( x_1 \) are the current and previous expansion points. Prior to the availability of two expansion points, a first-order Taylor series is used.

**actual_model_pointer**

- **Keywords Area**
- **model**
- **surrogate**
- **multipoint**
- **actual_model_pointer**

Pointer to specify a “truth” model, from which to construct a surrogate

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

**Alias:** none

**Argument(s):** STRING

**Description**

This must point to a model block, identified by \( \text{id\_model} \). That model will be run to generate training data, from which a surrogate model will be constructed.

See **block_pointer** for details about pointers.

**local**

- **Keywords Area**
- **model**
- **surrogate**
- **local**

Build a locally accurate surrogate from data at a single point

**Specification**

**Alias:** none

**Argument(s):** none
### Description

Local approximations use value, gradient, and possibly Hessian data from a single point to form a series expansion for approximating data in the vicinity of this point.

The currently available local approximation is the `taylor_series` selection.

The truth model to be used to generate the value/gradient/Hessian data used in the series expansion is identified through the required `actual_model_pointer` specification. The use of a model pointer (as opposed to an interface pointer) allows additional flexibility in defining the approximation. In particular, the derivative specification for the truth model may differ from the derivative specification for the approximation, and the truth model results being approximated may involve a model recursion (e.g., the values/gradients from a nested model).

### See Also

These keywords may also be of interest:

- global
- hierarchical
- multipoint

**taylor_series**

- Keywords Area
- model
- surrogate
- local
- `taylor_series`

Construct a Taylor Series expansion around a point

### Specification

**Alias:** none

**Argument(s):** none
6.3. MODEL

Description

The Taylor series model is purely a local approximation method. That is, it provides local trends in the vicinity of a single point in parameter space.

The order of the Taylor series may be either first-order or second-order, which is automatically determined from the gradient and Hessian specifications in the responses specification (see responses for info on how to specify gradients and Hessians) for the truth model.

Theory

The first-order Taylor series expansion is:

\[
\hat{f}(x) \approx f(x_0) + \nabla_x f(x_0)^T(x - x_0)
\]  

and the second-order expansion is:

\[
\hat{f}(x) \approx f(x_0) + \nabla_x f(x_0)^T(x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2_x f(x_0)(x - x_0)
\]  

where \(x_0\) is the expansion point in \(n\)-dimensional parameter space and \(f(x_0), \nabla_x f(x_0),\) and \(\nabla^2_x f(x_0)\) are the computed response value, gradient, and Hessian at the expansion point, respectively.

As dictated by the responses specification used in building the local surrogate, the gradient may be analytic or numerical and the Hessian may be analytic, numerical, or based on quasi-Newton secant updates.

In general, the Taylor series model is accurate only in the region of parameter space that is close to \(x_0\).

While the accuracy is limited, the first-order Taylor series model reproduces the correct value and gradient at the point \(x_0\), and the second-order Taylor series model reproduces the correct value, gradient, and Hessian. This consistency is useful in provably-convergent surrogate-based optimization. The other surface fitting methods do not use gradient information directly in their models, and these methods rely on an external correction procedure in order to satisfy the consistency requirements of provably-convergent SBO.

actual_model_pointer

- Keywords Area
- model
- surrogate
- local
- actual_model_pointer

Pointer to specify a "truth" model, from which to construct a surrogate

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING
CHAPTER 6. KEYWORDS AREA

Description

This must point to a model block, identified by \textit{id\_model}. That model will be run to generate training data, from which a surrogate model will be constructed.

See \textit{block\_pointer} for details about pointers.

Hierarchical

- Keywords Area
- model
- surrogate
- hierarchical

Hierarchical approximations use corrected results from a low fidelity model as an approximation to the results of a high fidelity “truth” model.

Specification

Alias: none

\textbf{Argument(s):} none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>\textit{low_fidelity_model_pointer}</td>
<td>Pointer to low fidelity model</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>\textit{high_fidelity_model_pointer}</td>
<td>Pointer to high fidelity model</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>\textit{correction}</td>
<td>Correction approaches for surrogate models</td>
</tr>
</tbody>
</table>

Description

Hierarchical approximations use corrected results from a low fidelity model as an approximation to the results of a high fidelity “truth” model. These approximations are also known as model hierarchy, multifidelity, variable fidelity, and variable complexity approximations. The required \textit{low\_fidelity\_model\_pointer} specification points to the low fidelity model specification. This model is used to generate low fidelity responses which are then corrected and returned to an iterator. The required \textit{high\_fidelity\_model\_pointer} specification points to the specification for the high fidelity truth model. This model is used only for verifying low fidelity results and updating low fidelity corrections. The \textit{correction} specification specifies which correction technique will be applied to the low fidelity results in order to match the high fidelity results at one or more points. In the hierarchical case (as compared to the global case), the \textit{correction} specification is required, since the omission of a correction technique would effectively eliminate the purpose of the high fidelity model. If it is desired to use a low fidelity model without corrections, then a hierarchical approximation is not needed and a single model should be used. Refer to \textit{global} for additional information on available correction approaches.
Theory

Multifidelity Surrogates: Multifidelity modeling involves the use of a low-fidelity physics-based model as a surrogate for the original high-fidelity model. The low-fidelity model typically involves a coarser mesh, looser convergence tolerances, reduced element order, or omitted physics. It is a separate model in its own right and does not require data from the high-fidelity model for construction. Rather, the primary need for high-fidelity evaluations is for defining correction functions that are applied to the low-fidelity results.

Multifidelity Surrogate Models

A second type of surrogate is the \{model hierarchy\} type (also called multifidelity, variable fidelity, variable complexity, etc.). In this case, a model that is still physics-based but is of lower fidelity (e.g., coarser discretization, reduced element order, looser convergence tolerances, omitted physics) is used as the surrogate in place of the high-fidelity model. For example, an inviscid, incompressible Euler CFD model on a coarse discretization could be used as a low-fidelity surrogate for a high-fidelity Navier-Stokes model on a fine discretization.

See Also

These keywords may also be of interest:

- global
- local
- multipoint

low_fidelity_model_pointer

- Keywords Area
- model
- surrogate
- hierarchical
- low_fidelity_model_pointer

Pointer to low fidelity model

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

Description

`low_fidelity_model_pointer` points to the model (using its `id_model` label) to use for generating low fidelity responses, which are corrected and returned to an iterator as explained on the parent page.
**high_fidelity_model_pointer**

- **Keywords Area**
- **model**
- **surrogate**
- **hierarchical**
- **high_fidelity_model_pointer**

Pointer to high fidelity model

**Topics**

This keyword is related to the topics:

- **block_pointer**

**Specification**

Alias: none  
**Argument(s):** STRING

**Description**

`high_fidelity_model_pointer` points to the model (using its `id_model` label) to use for verifying low fidelity results and updating low fidelity corrections, as explained on the parent page.

**correction**

- **Keywords Area**
- **model**
- **surrogate**
- **hierarchical**
- **correction**

Correction approaches for surrogate models

**Specification**

Alias: none  
**Argument(s):** none  
**Default:** no surrogate correction

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Description

Some of the surrogate model types support the use of correction factors that improve the local accuracy of the surrogate models.

The correction specification specifies that the approximation will be corrected to match truth data, either matching truth values in the case of zeroth-order matching, matching truth values and gradients in the case of first-order matching, or matching truth values, gradients, and Hessians in the case of second-order matching. For additive and multiplicative corrections, the correction is local in that the truth data is matched at a single point, typically the center of the approximation region. The additive correction adds a scalar offset (zeroth-order), a linear function (first-order), or a quadratic function (second-order) to the approximation to match the truth data at the point, and the multiplicative correction multiplies the approximation by a scalar (zeroth-order), a linear function (first-order), or a quadratic function (second-order) to match the truth data at the point. The additive first-order case is due to [57] and the multiplicative first-order case is commonly known as beta correction [40]. For the combined correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The combined correction is then a multipoint correction, as opposed to the local additive and multiplicative corrections. Each of these correction capabilities is described in detail in [24].

The correction factors force the surrogate models to match the true function values and possibly true function derivatives at the center point of each trust region. Currently, Dakota supports either zeroth-, first-, or second-order accurate correction methods, each of which can be applied using either an additive, multiplicative, or combined correction.

#### Correction Specification

<table>
<thead>
<tr>
<th>Required (Choose One)</th>
<th>correction order (Group 1)</th>
<th>zeroth_order</th>
<th>Specify that truth values must be matched.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>first_order</td>
<td>Specify that truth values and gradients must be matched.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>second_order</td>
<td>Specify that truth values, gradients and Hessians must be matched.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required (Choose One)</th>
<th>correction type (Group 2)</th>
<th>additive</th>
<th>Additive correction factor for local surrogate accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>multiplicative</td>
<td>Multiplicative correction factor for local surrogate accuracy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined</td>
<td>Multipoint correction for a hierarchical surrogate</td>
</tr>
</tbody>
</table>
correction function. For each of these correction approaches, the correction is applied to the surrogate model and
the corrected model is then interfaced with whatever algorithm is being employed. The default behavior is that no
correction factor is applied.

The simplest correction approaches are those that enforce consistency in function values between the surrogate
and original models at a single point in parameter space through use of a simple scalar offset or scaling applied
to the surrogate model. First-order corrections such as the first-order multiplicative correction (also known as
beta correction[15]) and the first-order additive correction[57] also enforce consistency in the gradients and pro-
vide a much more substantial correction capability that is sufficient for ensuring provable convergence in SBO
algorithms. SBO convergence rates can be further accelerated through the use of second-order corrections which
also enforce consistency in the Hessians[24], where the second-order information may involve analytic, finite-
difference, or quasi-Newton Hessians.

Correcting surrogate models with additive corrections involves

\[ \hat{f}_{hi}(x) = f_{lo}(x) + \alpha(x) \] (6.9)

where multifidelity notation has been adopted for clarity. For multiplicative approaches, corrections take the form

\[ \hat{f}_{hi}(x) = f_{lo}(x) \beta(x) \] (6.10)

where, for local corrections, \( \alpha(x) \) and \( \beta(x) \) are first or second-order Taylor series approximations to the exact
correction functions:

\[ \alpha(x) = A(x_c) + \nabla A(x_c)^T (x - x_c) + \frac{1}{2} (x - x_c)^T \nabla^2 A(x_c) (x - x_c) \] (6.11)

\[ \beta(x) = B(x_c) + \nabla B(x_c)^T (x - x_c) + \frac{1}{2} (x - x_c)^T \nabla^2 B(x_c) (x - x_c) \] (6.12)

where the exact correction functions are

\[ A(x) = f_{hi}(x) - f_{lo}(x) \] (6.13)

\[ B(x) = \frac{f_{hi}(x)}{f_{lo}(x)} \] (6.14)

Refer to[24] for additional details on the derivations.

A combination of additive and multiplicative corrections can provide for additional flexibility in minimizing
the impact of the correction away from the trust region center. In other words, both additive and multiplicative
corrections can satisfy local consistency, but through the combination, global accuracy can be addressed as well.
This involves a convex combination of the additive and multiplicative corrections:

\[ f_{hi}(x) = \gamma \hat{f}_{hi}(x) + (1 - \gamma) \hat{f}_{hi}(x) \]

where \( \gamma \) is calculated to satisfy an additional matching condition, such as matching values at the previous design
iterate.

It should be noted that in both first order correction methods, the function \( \hat{f}(x) \) matches the function value and
gradients of \( f(x) \) at \( x = x_c \). This property is necessary in proving that the first order-corrected SBO algorithms
are provably convergent to a local minimum of \( f(x) \). However, the first order correction methods are significantly
more expensive than the zeroth order correction methods, since the first order methods require computing both
\( \nabla f(x_c) \) and \( \nabla f(x_c) \). When the SBO strategy is used with either of the zeroth order correction methods, or
with no correction method, convergence is not guaranteed to a local minimum of \( f(x) \). That is, the SBO strategy
becomes a heuristic optimization algorithm. From a mathematical point of view this is undesirable, but as a
practical matter, the heuristic variants of SBO are often effective in finding local minima.

Usage guidelines
• Both the additive zeroth order and multiplicative zeroth order correction methods are "free" since they use values of $f_i(x_c)$ that are normally computed by the SBO strategy.

• The use of either the additive first order method or the multiplicative first order method does not necessarily improve the rate of convergence of the SBO algorithm.

• When using the first order correction methods, the gradient-related response keywords must be modified to allow either analytic or numerical gradients to be computed. This provides the gradient data needed to compute the correction function.

• For many computationally expensive engineering optimization problems, gradients often are too expensive to obtain or are discontinuous (or may not exist at all). In such cases the heuristic SBO algorithm has been an effective approach at identifying optimal designs[35].

zeroth order

• Keywords Area
  • model
  • surrogate
  • hierarchical
  • correction
  • zeroth order

Specify that truth values must be matched.

Specification

Alias: none
Argument(s): none

Description

The correction specification specifies that the approximation will be corrected to match truth data. The keyword zeroth order matching ensures that truth values are matched.

first order

• Keywords Area
  • model
  • surrogate
  • hierarchical
  • correction
  • first order

Specify that truth values and gradients must be matched.
**Specification**

*Alias:* none  
*Argument(s):* none

**Description**

This correction specification specifies that the approximation will be corrected to match truth data. The keyword `first_order` matching ensures that truth values and gradients are matched.

- **second_order**
  - `Keywords Area`
  - `model`
  - `surrogate`
  - `hierarchical`
  - `correction`
  - `second_order`

Specify that truth values, gradients and Hessians must be matched.

**Specification**

*Alias:* none  
*Argument(s):* none

**Description**

The correction specification specifies that the approximation will be corrected to match truth data. The keyword `second_order` matching ensures that truth values, gradients and Hessians are matched.

- **additive**
  - `Keywords Area`
  - `model`
  - `surrogate`
  - `hierarchical`
  - `correction`
  - `additive`

Additive correction factor for local surrogate accuracy

**Specification**

*Alias:* none  
*Argument(s):* none
Description
Use an additive correction factor to improve the local accuracy of a surrogate.

\textbf{multiplicative}

- Keywords Area
- model
- surrogate
- hierarchical
- correction
- multiplicative

Multiplicative correction factor for local surrogate accuracy.

Specification
\textbf{Alias:} none
\textbf{Argument(s):} none

Description
Use a multiplicative correction factor to improve the local accuracy of a surrogate.

\textbf{combined}

- Keywords Area
- model
- surrogate
- hierarchical
- correction
- combined

Multipoint correction for a hierarchical surrogate

Specification
\textbf{Alias:} none
\textbf{Argument(s):} none

Description
For the combined correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The combined correction is then a multipoint correction, as opposed to the local additive and multiplicative corrections.
6.3.7 nested

- Keywords Area
- model
- nested

A model whose responses are computed through the use of a sub-iterator

### Specification

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>optional_interface_pointer</td>
<td>Pointer to interface that provides non-nested responses</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>sub_method_pointer</td>
<td>The sub_method_pointer specifies the method block for the sub-iterator</td>
</tr>
</tbody>
</table>

### Description

Instead of appealing directly to a primary interface, a nested model maps variables to responses by executing a secondary iterator, or a "sub-iterator". In other words, a function evaluation of the primary study consists of a solution of an entire secondary study - potentially many secondary function evaluations.

The sub-iterator in turn operates on a sub-model. The sub-iterator responses may be combined with non-nested contributions from an optional interface specification.

A sub_method_pointer must be provided in order to specify the method block describing the sub-iterator. The remainder of the model is specified under that keyword.

A optional_interface_pointer points to the interface specification and optional_interface_responses_pointer points to a responses specification describing the data to be returned by this interface). This interface is used to provide non-nested data, which is then combined with data from the nested iterator using the primary_response_mapping and secondary_response_mapping inputs (see mapping discussion below).

### Examples

An example of variable and response mappings is provided below:

```plaintext
primary_variable_mapping = '' '' 'X' 'Y'
secondary_variable_mapping = '' '' 'mean' 'mean'
primary_response_mapping = 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
secondary_response_mapping = 0.0 0.0 1.3 0.0 0.0 0.0 0.0 1.3 0.
```
The variable mappings correspond to 4 top-level variables, the first two of which employ the default mappings from active top-level variables to sub-model variables of the same type (option 3 above) and the latter two of which are inserted into the mean distribution parameters of sub-model variables \( \bar{X} \) and \( \bar{Y} \) (option 1 above). The response mappings define a 3 by 9 matrix corresponding to 9 inner loop response attributes and 3 outer loop response functions (one primary response function and 2 secondary functions, such as one objective and two constraints). Each row of the response mapping is a vector which is multiplied (i.e., with a dot-product) against the 9 sub-iterator values to determine the outer loop function. Consider a UQ example with 3 response functions, each providing a mean, a standard deviation, and one level mapping (if no level mappings are specified, the responses would only have a mean and standard deviation). The primary response mapping can be seen to extract the first value from the inner loop, which would correspond to the mean of the first response function. This mapped sub-iterator response becomes a single objective function, least squares term, or generic response function at the outer level, as dictated by the top-level response specification. The secondary response mapping maps the fourth sub-iterator response function plus 3 times the fifth sub-iterator response function (mean plus 3 standard deviations) into one top-level nonlinear constraint and the seventh sub-iterator response function plus 3 times the eighth sub-iterator response function (mean plus 3 standard deviations) into another top-level nonlinear constraint, where these top-level nonlinear constraints may be inequality or equality, as dictated by the top-level response specification. Note that a common case is for each sub-iterator response to be mapped to a unique outer loop response (for example, in the nested UQ case where one wants to determine an interval on each inner loop statistic). In these simple cases, the response mapping would define an identity matrix.

**See Also**

These keywords may also be of interest:

- single
- surrogate

**optional_interface_pointer**

- Keywords Area
- model
- nested
- optional_interface_pointer

Pointer to interface that provides non-nested responses

**Topics**

This keyword is related to the topics:

- block_pointer

**Specification**

- **Alias:** none
  - **Argument(s):** STRING
  - **Default:** no optional interface
### Description

`optional_interface_pointer` is used to specify an optional interface (using that interface block’s `id_interface` label) to provide non-nested responses, which will be combined with responses from the nested sub-iterator. The `primary_response_mapping` and `secondary_response_mapping` keywords control how responses are combined.

**optional_interface_responses_pointer**

- **Keywords Area**
- **model**
- **nested**
- **optional_interface_pointer**
- **optional_interface_responses_pointer**

Pointer to responses block that defines non-nested responses

### Topics

This keyword is related to the topics:

- **block_pointer**

### Specification

**Alias:** none

**Argument(s):** STRING

**Default:** reuse of top-level responses specification

### Description

`optional_interface_responses_pointer` points to the responses block (specifically, its `id_responses` label) that defines the non-nested response to return to the nested model. The `primary_response_mapping` and `secondary_response_mapping` keywords control how these non-nested responses are combined with responses from the nested sub-iterator. If `optional_interface_responses_pointer` is not provided, the top-level responses specification is reused.
6.3. MODEL

sub_method_pointer

- Keywords Area
- model
- nested
- sub_method_pointer

The sub_method_pointer specifies the method block for the sub-iterator

Topics

This keyword is related to the topics:

- block_pointer

Specification

Alias: none

Argument(s): STRING

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>iterator_servers</td>
<td>Specify the number of iterator servers when Dakota is run in parallel</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>iterator_scheduling</td>
<td>Specify the scheduling of concurrent iterators when Dakota is run in parallel</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>processors_per_iter</td>
<td>Specify the number of processors per iterator server when Dakota is run in parallel</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>primary_variable_mapping</td>
<td>Primary mapping of top-level variables to sub-model variables</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Optional

- secondary_variable_mapping

Optional

- primary_response_mapping

Optional

- secondary_response_mapping

Description

The sub_method_pointer specifies the method block for the sub-iterator.

See block_pointer for details about pointers.

Nested models may employ mappings for both the variable inputs to the sub-model and the response outputs from the sub-model. In the former case, the primary_variable_mapping and secondary_variable_mapping specifications are used to map from the active top-level variables into the sub-model variables, and in the latter case, the primary_response_mapping and secondary_response_mapping specifications are used to compute the sub-model response contributions to the top-level responses.

For the variable mappings, the primary and secondary specifications provide lists of strings which are used to target specific sub-model variables and their sub-parameters, respectively. The primary strings are matched to continuous or discrete variable labels such as ‘cdv_1’ (either user-supplied or default labels), and the secondary strings are matched to either real or integer random variable distribution parameters such as ‘mean’ or ‘num_trials’ (the form of the uncertain distribution parameter keyword that is appropriate for a single variable instance) or continuous or discrete design/state variable sub-parameters such as ‘lower_bound’ or ‘upper_bound’ (again, keyword form appropriate for a single variable instance). No coercion of types is supported, so real-valued top-level variables should map to either real-valued sub-model variables or real-valued sub-parameters and integer-valued top-level variables should map to either integer-valued sub-model variables or integer-valued sub-parameters. As long as these real versus integer constraints are satisfied, mappings are free to cross variable types (design, aleatory uncertain, epistemic uncertain, state) and domain types (continuous, discrete). Both primary_variable_mapping and secondary_variable_mapping specifications are optional, which is designed to support the following three possibilities:

1. If both primary and secondary variable mappings are specified, then an active top-level variable value will be inserted into the identified sub-parameter (the secondary mapping) for the identified sub-model variable (the primary mapping).

2. If a primary mapping is specified but a secondary mapping is not, then an active top-level variable value will be inserted into the identified sub-model variable value (the primary mapping).

3. If a primary mapping is not specified (corresponding secondary mappings, if specified, are ignored), then an active top-level variable value will be inserted into a corresponding sub-model variable, based on matching of variable types (e.g., top-level and sub-model variable specifications both allocate a set of ‘continuous_design’ variables which are active at the top level). Multiple sub-model variable types may be updated in this manner, provided that they are all active in the top-level variables. Since there is a direct variable correspondence for these default insertions, sub-model bounds and labels are also updated.
6.3. **MODEL**

from the top-level bounds and labels in order to eliminate the need for redundant input file specifications. Thus, it is typical for the sub-model variables specification to only contain the minimal required information, such as the number of variables, for these insertion targets. The sub-model must allocate enough space for each of the types that will accept default insertions, and the leading set of matching sub-model variables are updated (i.e., the sub-model may allocate more than needed and the trailing set will be unmodified).

These different variable mapping possibilities may be used in any combination by employing empty strings ("") for particular omitted mappings (the number of strings in user-supplied primary and secondary variable mapping specifications must equal the total number of active top-level variables, including both continuous and discrete types). The ordering of the active variables is the same as shown in dakota.input.summary on Input Spec Summary and as presented in **variables**.

If inactive variables are present at the outer level, then the default type 3 mapping is used for these variables; that is, outer loop inactive variables are inserted into inner loop variables (active or inactive) based on matching of variable types, top-level bounds and labels are also propagated, the inner loop must allocate sufficient space to receive the outer loop values, and the leading subset within this inner loop allocation is updated. This capability is important for allowing nesting beyond two levels, since an active variable at the outer-most loop may become inactive at the next lower level, but still needs to be further propagated down to lower levels in the recursion.

For the response mappings, the primary and secondary specifications provide real-valued multipliers to be applied to sub-iterator response results so that the responses from the inner loop can be mapped into a new set of responses at the outer loop. For example, if the nested model is being employed within a mixed aleatory-epistemic uncertainty quantification, then aleatory statistics from the inner loop (such as moments of the response) are mapped to the outer level, where minima and maxima of these aleatory statistics are computed as functions of the epistemic parameters. The response mapping defines a matrix which scales the values from the inner loop and determines their position in the outer loop response vector. Each row of the mapping corresponds to one outer loop response, where each column of the mapping corresponds to a value from the inner loop. Depending on the number of responses and the particular attributes calculated on the inner loop, there will be a vector of inner loop response values that need to be accounted for in the mapping. This vector of inner loop response results is defined as follows for different sub-iterator types:

- **optimization**: the final objective function(s) and nonlinear constraints
- **nonlinear least squares**: the final least squares terms and nonlinear constraints
- **aleatory uncertainty quantification (UQ)**: for each response function, a mean statistic, a standard deviation statistic, and all probability/reliability/generalized reliability/response level results for any user-specified response_levels, probability_levels, reliability_levels, and/or gen_reliability_levels, in that order.
- **epistemic and mixed aleatory/epistemic UQ using interval estimation methods**: lower and upper interval bounds for each response function.
- **epistemic and mixed aleatory/epistemic UQ using evidence methods**: for each response function, lower and upper interval bounds (belief and plausibility) for all probability/reliability/generalized reliability/response level results computed from any user-specified response_levels, probability_levels, reliability_levels, and/or gen_reliability_levels, in that order.
- **parameter studies and design of experiments**: for optimization and least squares response data sets, the best solution found (lowest constraint violation if infeasible, lowest composite objective function if feasible). For generic response data sets, a best solution metric is not defined, so the sub-iterator response vector is empty in this case.
The primary values map sub-iterator response results into top-level objective functions, least squares terms, or generic response functions, depending on the declared top-level response set. The secondary values map sub-iterator response results into top-level nonlinear inequality and equality constraints.

Nested models utilize a sub-iterator and a sub-model to perform a complete iterative study as part of every evaluation of the model. This sub-iteration accepts variables from the outer level, performs the sub-level analysis, and computes a set of sub-level responses which are passed back up to the outer level. Mappings are employed for both the variable inputs to the sub-model and the response outputs from the sub-model.

In the variable mapping case, primary and secondary variable mapping specifications are used to map from the top-level variables into the sub-model variables. These mappings support three possibilities in any combination: (1) insertion of an active top-level variable value into an identified sub-model distribution parameter for an identified active sub-model variable, (2) insertion of an active top-level variable value into an identified active sub-model variable value, and (3) addition of an active top-level variable value as an inactive sub-model variable, augmenting the active sub-model variables.

In the response mapping case, primary and secondary response mapping specifications are used to map from the sub-model responses back to the top-level responses. These specifications provide real-valued multipliers that are applied to the sub-iterator response results to define the outer level response set. These nested data results may be combined with non-nested data through use of the "optional interface" component within nested models.

The nested model constructs admit a wide variety of multi-iterator, multi-model solution approaches. For example, optimization within optimization (for hierarchical multidisciplinary optimization), uncertainty quantification within uncertainty quantification (for second-order probability), uncertainty quantification within optimization (for optimization under uncertainty), and optimization within uncertainty quantification (for uncertainty of optima) are all supported, with and without surrogate model indirection. Several examples of nested model usage are provided in the Users Manual, most notably mixed epistemic-aleatory UQ, optimization under uncertainty (OUU), and surrogate-based UQ.

**iterator_servers**

- **Keywords Area**
- **model**
- **nested**
- **sub_method_pointer**
- **iterator_servers**

Specify the number of iterator servers when Dakota is run in parallel

**Topics**

This keyword is related to the topics:

- **concurrency_and_parallelism**

**Specification**

**Alias**: none

**Argument(s)**: INTEGER
Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional iterator_servers specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

iterator_scheduling

- Keywords Area
- model
- nested
- sub_method_pointer
- iterator_scheduling

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional Required Group</th>
<th>Description Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td></td>
<td>peer</td>
<td>Specify a peer partition for parallel iterator scheduling</td>
</tr>
</tbody>
</table>

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional iterator_scheduling specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, hybrid, multi_start, and pareto_set component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.
master

Specify a dedicated master partition for parallel iterator scheduling

Topics
This keyword is related to the topics:
  - concurrency_and_parallelism

Specification
Alias: none
Argument(s): none

Description
This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

peer

Specify a peer partition for parallel iterator scheduling

Topics
This keyword is related to the topics:
  - concurrency_and_parallelism
6.3. **MODEL**

### Specification

**Alias:** none

**Argument(s):** none

### Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of evaluation-scheduling, it is not possible to specify static or dynamic.

**processors_per_iterator**

- **Keywords Area**
- **model**
- **nested**
- **sub_method_pointer**
- **processors_per_iterator**

Specify the number of processors per iterator server when Dakota is run in parallel.

### Topics

This keyword is related to the topics:

- concurrency and parallelism

### Specification

**Alias:** none

**Argument(s):** INTEGER

### Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterations. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual[4] for additional information.

**primary_variable_mapping**

- **Keywords Area**
- **model**
- **nested**
CHAPTER 6. KEYWORDS AREA

- sub_method_pointer

- primary_variable_mapping

  Primary mapping of top-level variables to sub-model variables

**Specification**

**Alias:** none

  **Argument(s):** STRINGLIST
  **Default:** default variable insertions based on variable type

**Description**

The `primary_variable_mapping`, `secondary_variable_mapping`, `primary_response_mapping`, and `secondary_response_mapping` keywords control how top-level variables and responses are mapped to variables and responses in the sub-model. Their usage is explained on the parent keyword (sub_method_pointer) page.

**secondary_variable_mapping**

- Keywords Area

- model

- nested

- sub_method_pointer

- secondary_variable_mapping

  Secondary mapping of top-level variables to sub-model variables

**Specification**

**Alias:** none

  **Argument(s):** STRINGLIST
  **Default:** primary mappings into sub-model variables are value-based

**Description**

The `primary_variable_mapping`, `secondary_variable_mapping`, `primary_response_mapping`, and `secondary_response_mapping` keywords control how top-level variables and responses are mapped to variables and responses in the sub-model. Their usage is explained on the parent keyword (sub_method_pointer) page.
6.3. **MODEL**

**primary_response_mapping**

- **Keywords Area**
- **model**
- **nested**
- **sub_method_pointer**
- **primary_response_mapping**

Primary mapping of sub-model responses to top-level responses

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** no sub-iterator contribution to primary functions

**Description**

The `primary_variable_mapping, secondary_variable_mapping, primary_response_mapping, and secondary_response_mapping` keywords control how top-level variables and responses are mapped to variables and responses in the sub-model. Their usage is explained on the parent keyword (`sub_method_pointer`) page.

**secondary_response_mapping**

- **Keywords Area**
- **model**
- **nested**
- **sub_method_pointer**
- **secondary_response_mapping**

Secondary mapping of sub-model responses to top-level responses

**Specification**

**Alias:** none  
**Argument(s):** REALLIST  
**Default:** no sub-iterator contribution to secondary functions

**Description**

The `primary_variable_mapping, secondary_variable_mapping, primary_response_mapping, and secondary_response_mapping` keywords control how top-level variables and responses are mapped to variables and responses in the sub-model. Their usage is explained on the parent keyword (`sub_method_pointer`) page.
6.4 variables

- Keywords Area
- variables

Specifies the parameter set to be iterated by a particular method.

Topics

This keyword is related to the topics:

- block

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>id_variables</td>
<td></td>
<td>Name the variables block; helpful when there are multiple</td>
</tr>
<tr>
<td>Optional</td>
<td>active</td>
<td></td>
<td>Set the active variables view a method will see</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>mixed</td>
<td></td>
<td>Maintain continuous/discrete variable distinction</td>
</tr>
<tr>
<td>Optional</td>
<td>relaxed</td>
<td></td>
<td>Allow treatment of discrete variables as continuous</td>
</tr>
<tr>
<td>Optional</td>
<td>continuous_design</td>
<td></td>
<td>Continuous design variables; each defined by a real interval</td>
</tr>
<tr>
<td>Optional</td>
<td>discrete_design-range</td>
<td>Discrete design variables; each defined by an integer interval</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>discrete_design_set</td>
<td>Set-valued discrete design variables</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>normal_uncertain</td>
<td>Aleatory uncertain variable - normal (Gaussian)</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>lognormal_uncertain</td>
<td>Aleatory uncertain variable - lognormal</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>uniform_uncertain</td>
<td>Aleatory uncertain variable - uniform</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>loguniform_uncertain</td>
<td>Aleatory uncertain variable - loguniform</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>triangular_uncertain</td>
<td>Aleatory uncertain variable - triangular</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>exponential_uncertain</td>
<td>Aleatory uncertain variable - exponential</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>beta_uncertain</td>
<td>Aleatory uncertain variable - beta</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>gamma_uncertain</td>
<td>Aleatory uncertain variable - gamma</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>gumbel_uncertain</td>
<td>Aleatory uncertain variable - gumbel</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>frechet_uncertain</td>
<td>Aleatory uncertain variable - Frechet</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>weibull_uncertain</td>
<td>Aleatory uncertain variable - Weibull</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>histogram_bin_uncertain</td>
<td>Aleatory uncertain variable - continuous histogram</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>poisson_uncertain</td>
<td>Aleatory uncertain discrete variable - Poisson</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------</td>
<td>------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>binomial_uncertain</td>
<td>Aleatory uncertain discrete variable - binomial</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>negative_binomial_uncertain</td>
<td>Aleatory uncertain discrete variable - negative binomial</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>geometric_uncertain</td>
<td>Aleatory uncertain discrete variable - geometric</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>hypergeometric_uncertain</td>
<td>Aleatory uncertain discrete variable - hypergeometric</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>histogram_point_uncertain</td>
<td>Correlation among aleatory uncertain variables</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>uncertain_correlation_matrix</td>
<td>Epistemic uncertain variable - values from one or more continuous intervals</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>continuous_interval_uncertain</td>
<td>Epistemic uncertain variable - values from one or more discrete intervals</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>discrete_interval_uncertain</td>
<td>Set-valued discrete uncertain variables</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>discrete_uncertain_set</td>
<td>Continuous state variables</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>continuous_state</td>
<td>Discrete state variables; each defined by an integer interval</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>discrete_state_range</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.4. VARIABLES

<table>
<thead>
<tr>
<th>Optional</th>
<th>discrete_state_set</th>
<th>Set-valued discrete state variables</th>
</tr>
</thead>
</table>

**Description**

The variables specification in a Dakota input file specifies the parameter set to be iterated by a particular method.

In the case of

- An optimization study:
  - These variables are adjusted in order to locate an optimal design.

- Parameter studies/sensitivity analysis/design of experiments:
  - These parameters are perturbed to explore the parameter space.

- Uncertainty analysis:
  - The variables are associated with distribution/interval characterizations which are used to compute corresponding distribution/interval characterizations for response functions.

To accommodate these different studies, Dakota supports different:

- Variable types
  - design
  - aleatory uncertain
  - epistemic uncertain
  - state

- Variable domains
  - continuous
  - discrete
    - discrete range
    - discrete integer set
    - discrete string set
    - discrete real set

Use the variables page to browse the available variables by type and domain.

**Variable Types**

- Design Variables
  - Design variables are those variables which are modified for the purposes of computing an optimal design.
  - The most common type of design variables encountered in engineering applications are of the continuous type. These variables may assume any real value within their bounds.
  - All but a handful of the optimization algorithms in Dakota support continuous design variables exclusively.
• Aleatory Uncertain Variables
  – Aleatory uncertainty is also known as inherent variability, irreducible uncertainty, or randomness.
  – Aleatory uncertainty is predominantly characterized using probability theory. This is the only option implemented in Dakota.

• Epistemic Uncertain Variables
  – Epistemic uncertainty is uncertainty due to lack of knowledge.
  – In Dakota, epistemic uncertainty is characterized by interval analysis or the Dempster-Shafer theory of evidence.
  – Note that epistemic uncertainty can also be modeled with probability density functions (as done with aleatory uncertainty). Dakota does not support this capability.

• State Variables
  – State variables consist of "other" variables which are to be mapped through the simulation interface, in that they are not to be used for design and they are not modeled as being uncertain.
  – State variables provide a convenient mechanism for managing additional model parameterizations such as mesh density, simulation convergence tolerances, and time step controls.
  – Only parameter studies and design of experiments methods will iterate on state variables.
  – The _initial_value_ is used as the only value for the state variable for all other methods, unless _active state_ is invoked.
  – See more details on the state variables page.

**Variable Domains**
Continuous variables are defined by bounds. Discrete variables can be defined in one of three ways, which are discussed on the page _discrete variables_.

**Ordering of Variables**
The ordering of variables is important, and a consistent ordering is employed throughout the Dakota software. The ordering is shown in _dakota.input.summary_ and can be summarized as:

1. **design**
   - (a) continuous
   - (b) discrete integer
   - (c) discrete string
   - (d) discrete real

2. **aleatory uncertain**
   - (a) continuous
   - (b) discrete integer
   - (c) discrete string
   - (d) discrete real

3. **epistemic uncertain**
   - (a) continuous
6.4. VARIABLES

- (b) discrete integer
- (c) discrete string
- (d) discrete real

4. state

- (a) continuous
- (b) discrete integer
- (c) discrete string
- (d) discrete real

Ordering of variable types below this granularity (e.g., from normal to histogram bin within aleatory uncertain - continuous) is defined somewhat arbitrarily, but is enforced consistently throughout the code.

**Active Variables**

The reason variable types exist is that methods have the capability to treat variable types differently. All methods have default behavior that determines which variable types are "active" and will be assigned values by the method. For example, optimization methods will only vary the design variables - by default.

The default behavior should be described on each method page, or on topics pages that relate to classes of methods. In addition, the default behavior can be modified using the `active` keyword.

**Examples**

Several examples follow. In the first example, two continuous design variables are specified:

```plaintext
variables,
  continuous_design = 2
  initial_point 0.9 1.1
  upper_bounds  5.8 2.9
  lower_bounds  0.5 -2.9
  descriptors 'radius' 'location'
```

In the next example, defaults are employed. In this case, `initial_point` will default to a vector of 0. values, `upper_bounds` will default to vector values of `DBL_MAX` (the maximum number representable in double precision for a particular platform, as defined in the platform's `float.h` C header file), `lower_bounds` will default to a vector of `-DBL_MAX` values, and `descriptors` will default to a vector of 'cdv_i' strings, where i ranges from one to two:

```plaintext
variables,
  continuous_design = 2
```

In the following example, the syntax for a normal-lognormal distribution is shown. One normal and one lognormal uncertain variable are completely specified by their means and standard deviations. In addition, the dependence structure between the two variables is specified using the `uncertain_correlation_matrix`.

```plaintext
variables,
  normal_uncertain = 1
  means = 1.0
  std_deviations = 1.0
  descriptors = 'TF1n'
  lognormal_uncertain = 1
  means = 2.0
  std_deviations = 0.5
  descriptors = 'TF2ln'
  uncertain_correlation_matrix = 1.0 0.2
                                 0.2 1.0
```
An example of the syntax for a state variables specification follows:

```plaintext
variables,
    continuous_state = 1
    initial_state 4.0
    lower_bounds 0.0
    upper_bounds 8.0
    descriptors 'CS1'
    discrete_state_range = 1
    initial_state 104
    lower_bounds 100
    upper_bounds 110
    descriptors 'DS1'
```

And in a more advanced example, a variables specification containing a set identifier, continuous and discrete design variables, normal and uniform uncertain variables, and continuous and discrete state variables is shown:

```plaintext
variables,
    id_variables = 'V1'
    continuous_design = 2
    initial_point 0.9 1.1
    upper_bounds 5.8 2.9
    lower_bounds 0.5 -2.9
    descriptors 'radius' 'location'
    discrete_design_range = 1
    initial_point 2
    upper_bounds 1
    lower_bounds 3
    descriptors 'material'
    normal_uncertain = 2
    means = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors = 'TF1n' 'TF2n'
    uniform_uncertain = 2
    lower_bounds = 199.3, 474.63
    upper_bounds = 298.5, 712.
    descriptors = 'TF1u' 'TF2u'
    continuous_state = 2
    initial_state = 1.e-4 1.e-6
    descriptors = 'EPSIT1' 'EPSIT2'
    discrete_state_set_int = 1
    initial_state = 100
    set_values = 100 212 375
    descriptors = 'load_case'
```

### 6.4.1 id_variables

- Keywords Area
- variables
- id_variables

Name the variables block; helpful when there are multiple

### Topics

This keyword is related to the topics:

- block_pointer
6.4. VARIABLES

Specification

Alias: none

- Argument(s): STRING
- Default: use of last variables parsed

Description

The optional set identifier specification uses the keyword \texttt{id.variables} to input a unique string for use in identifying a particular variables set. A model can then identify the use of this variables set by specifying the same string in its \texttt{variables.pointer} specification.

If the \texttt{id.variables} specification is omitted, a particular variables set will be used by a model only if that model omits specifying \texttt{variables.pointer} and if the variables set was the last set parsed (or is the only set parsed). In common practice, if only one variables set exists, then \texttt{id.variables} can be safely omitted from the variables specification and \texttt{variables.pointer} can be omitted from the model specification(s), since there is no potential for ambiguity in this case.

Examples

For example, a model whose specification contains \texttt{variables.pointer = 'V1'} will use a variables specification containing the set identifier \texttt{id.variables = 'V1'}.

See Also

These keywords may also be of interest:

- \texttt{variables.pointer}

6.4.2 active

- Keywords Area
- variables
- active

Set the active variables view a method will see

Specification

Alias: none

- Argument(s): none
- Default: Infer from response or method specification

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td></td>
<td>active keyword</td>
<td></td>
</tr>
</tbody>
</table>

Required (Choose One) Group 1
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>design</th>
<th>Option for the active keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>uncertain</td>
<td>Option for the active keyword</td>
</tr>
<tr>
<td>aleatory</td>
<td>Option for the active keyword</td>
</tr>
<tr>
<td>epistemic</td>
<td>Option for the active keyword</td>
</tr>
<tr>
<td>state</td>
<td>Option for the active keyword</td>
</tr>
</tbody>
</table>

### Description

There are certain situations where the user may want to explicitly control the subset of variables that is considered active for a certain Dakota method. This is done by specifying the keyword `active` in the variables specification block, followed by one of the following: `all`, `design`, `uncertain`, `aleatory`, `epistemic`, or `state`.

Specifying one of these subsets of variables will allow the Dakota method to operate on the specified variable types and override the default active subset.

If the user does not specify any explicit override of the active view of the variables, Dakota first considers the response function specification.

- If the user specifies objective functions or calibration terms in the response specification block, then we can infer that the active variables should be the design variables (since design variables are used within optimization and least squares methods).

- If the user instead specifies the generic response type of `response_functions`, then Dakota cannot infer the active variable subset from the response specification and will instead infer it from the method selection.

1. If the method is a parameter study, or any of the methods available under dace, psuade, or fsu methods, the active view is set to all variables.
2. For uncertainty quantification methods, if the method is sampling, then the view is set to aleatory if only aleatory variables are present, epistemic if only epistemic variables are present, or uncertain (covering both aleatory and epistemic) if both are present.
3. If the uncertainty method involves interval estimation or evidence calculations, the view is set to epistemic.
4. For other uncertainty quantification methods not mentioned in the previous sentences (e.g., reliability methods or stochastic expansion methods), the default view is set to aleatory.
5. Finally, for verification studies using the Richardson extrapolation method, the active view is set to state.
6. Note that in surrogate-based optimization, where the surrogate is built on points defined by the method defined by the `dace_method_pointer`, the sampling used to generate the points is performed only over the design variables as a default unless otherwise specified (e.g. state variables will not be sampled for surrogate construction).

As alluded to in the previous section, the iterative method selected for use in Dakota determines what subset, or view, of the variables data is active in the iteration. The general case of having a mixture of various different types of variables is supported within all of the Dakota methods even though certain methods will only modify certain types of variables (e.g., optimizers and least squares methods only modify design variables, and uncertainty quantification methods typically only utilize uncertain variables). This implies that variables which are not under
6.4. VARIABLES

the direct control of a particular iterator will be mapped through the interface in an unmodified state. This allows for a variety of parameterizations within the model in addition to those which are being used by a particular iterator, which can provide the convenience of consolidating the control over various modeling parameters in a single file (the Dakota input file). An important related point is that the variable set that is active with a particular iterator is the same variable set for which derivatives are typically computed.

Examples

For example, the default behavior for a nondeterministic sampling method is to sample the uncertain variables. However, if the user specified active all in the variables specification block, the sampling would be performed over all variables (e.g. design and state variables in addition to the uncertain variables). This may be desired in situations such as surrogate-based optimization under uncertainty, where a surrogate may be constructed to span both design and uncertain variables. This is an example where we expand the active subset beyond the default, but in other situations, we may wish to restrict from the default. An example of this would be performing design of experiments in the presence of multiple variable types (for which all types are active by default), but only wanting to sample over the design variables for purposes of constructing a surrogate model for optimization.

Theory

The optional status of the different variable type specifications allows the user to specify only those variables which are present (rather than explicitly specifying that the number of a particular type of variables is zero). However, at least one type of variables that are active for the iterator in use must have nonzero size or an input error message will result.

all

• Keywords Area
• variables
• active
• all

Option for the active keyword

Specification

Alias: none
Argument(s): none

Description

See the active keyword

design

• Keywords Area
• variables
• active
• design

Option for the active keyword

**Specification**

Alias: none
Argument(s): none

**Description**

See the active keyword

**uncertain**

• Keywords Area
  • variables
  • active
  • uncertain

Option for the active keyword

**Specification**

Alias: none
Argument(s): none

**Description**

See the active keyword

**aleatory**

• Keywords Area
  • variables
  • active
  • aleatory

Option for the active keyword

**Specification**

Alias: none
Argument(s): none

**Description**

See the active keyword
6.4. VARIABLES

epistemic
  • Keywords Area
  • variables
  • active
  • epistemic

Option for the active keyword

Specification
Alias: none
  Argument(s): none

Description
See the active keyword

state
  • Keywords Area
  • variables
  • active
  • state

Option for the active keyword

Specification
Alias: none
  Argument(s): none

Description
See the active keyword

6.4.3 mixed
  • Keywords Area
  • variables
  • mixed

Maintain continuous/discrete variable distinction
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none
Default: relaxed (branch and bound), mixed (all other methods)

Description

The variables domain specifies how the discrete variables are treated. If the user specifies mixed in the variable specification block, the continuous and discrete variables are treated separately. If the user specifies relaxed in the variable specification block, the discrete variables are relaxed and treated as continuous variables. This may be useful in optimization problems involving both continuous and discrete variables when a user would like to use an optimization method that is designed for continuous variable optimization. All Dakota methods have a default value of mixed for the domain type except for the branch-and-bound method which has a default domain type of relaxed. Note that the branch-and-bound method is under development at this time. Finally, note that the domain selection applies to all variable types: design, aleatory uncertain, epistemic uncertain, and state.

With respect to domain type, if the user does not specify an explicit override of mixed or relaxed, Dakota infers the domain type from the method. As mentioned above, all methods currently use a mixed domain as a default, except the branch-and-bound method which is under development.

See Also

These keywords may also be of interest:

- relaxed

6.4.4 relaxed

- Keywords Area
- variables
- relaxed

Allow treatment of discrete variables as continuous

Specification

Alias: none
Argument(s): none

Description

The variables domain specifies how the discrete variables are treated. If the user specifies mixed in the variable specification block, the continuous and discrete variables are treated separately. If the user specifies relaxed in the variable specification block, the discrete variables are relaxed and treated as continuous variables. This may be useful in optimization problems involving both continuous and discrete variables when a user would like to use an optimization method that is designed for continuous variable optimization. All Dakota methods have a default value of mixed for the domain type except for the branch-and-bound method which has a default domain type of relaxed. Note that the branch-and-bound method is under development at this time. Finally, note that the domain selection applies to all variable types: design, aleatory uncertain, epistemic uncertain, and state.
With respect to domain type, if the user does not specify an explicit override of `mixed` or `relaxed`, Dakota infers the domain type from the method. As mentioned above, all methods currently use a mixed domain as a default, except the branch-and-bound method which is under development.

See Also

These keywords may also be of interest:

- `mixed`

6.4.5 `continuous_design`

- **Keywords Area**
- `variables`
- `continuous_design`

Continuous design variables; each defined by a real interval

Topics

This keyword is related to the topics:

- `continuous_variables`
- `design_variables`

Specification

Alias: none

**Argument(s):** INTEGER

**Default:** no continuous design variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>lower_bounds</td>
<td></td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Optional</td>
<td>upper_bounds</td>
<td></td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td>scale_types</td>
<td></td>
<td>Specify scaling for the variables</td>
</tr>
<tr>
<td>Optional</td>
<td>scales</td>
<td></td>
<td>Specify scaling for the variables</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

Continuous variables that are changed during the search for the optimal design.
CHAPTER 6. KEYWORDS AREA

initial_point
- Keywords Area
- variables
- continuous_design
- initial_point

Initial values

Specification
Alias: cdv_initial_point
Argument(s): REALLIST
Default: 0.0

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

lower_bounds
- Keywords Area
- variables
- continuous_design
- lower_bounds

Specify minimum values

Specification
Alias: cdv_lower_bounds
Argument(s): REALLIST
Default: -infinity

Description
Specify minimum values

upper_bounds
- Keywords Area
- variables
- continuous_design
- upper_bounds

Specify maximum values
6.4. VARIABLES

**Specification**

Alias: cdv_upper_bounds  
**Argument(s):** REALLIST  
**Default:** infinity

**Description**

Specify maximum values

**scale_types**

- **Keywords Area**
- **variables**
- **continuous_design**
- **scale_types**

Specify scaling for the variables

**Specification**

Alias: cdv_scale_types  
**Argument(s):** STRINGLIST  
**Default:** vector values = ‘none’

**Description**

For continuous variables, the scale_types specification includes strings specifying the scaling type for each component of the continuous design variables vector in methods that support scaling, when scaling is enabled.

Each entry in scale_types may be selected from ‘none’, ‘value’, ‘auto’, or ‘log’, to select no, characteristic value, automatic, or logarithmic scaling, respectively. If a single string is specified it will apply to all components of the continuous design variables vector. Each entry in scales may be a user-specified nonzero real characteristic value to be used in scaling each variable component. These values are ignored for scaling type ‘none’, required for ‘value’, and optional for ‘auto’ and ‘log’. If a single real value is specified it will apply to all components of the continuous design variables vector.

**Examples**

Two continuous design variables, one scaled by a value, the other log scaled,

```
continuous_design = 2
initial_point   -1.2  1.0
lower_bounds   -2.0  0.001
upper_bounds   2.0  2.0
descriptors    'x1'   'x2'
scale_types     'value' 'log'
scales          4.0   0.1
```
CHAPTER 6. KEYWORDS AREA

scales

- Keywords Area
- variables
- continuous_design
- scales

Specify scaling for the variable.

Specification

**Alias:** cdv_scales

**Argument(s):** REALLIST

**Default:** vector values = 1. (no scaling)

Description

For continuous variables, the `scale_types` specification includes strings specifying the scaling type for each component of the continuous design variables vector in methods that support scaling, when scaling is enabled. Each entry in `scale_types` may be selected from 'none', 'value', 'auto', or 'log', to select no, characteristic value, automatic, or logarithmic scaling, respectively. If a single string is specified it will apply to all components of the continuous design variables vector. Each entry in `scales` may be a user-specified nonzero real characteristic value to be used in scaling each variable component. These values are ignored for scaling type 'none', required for 'value', and optional for 'auto' and 'log'. If a single real value is specified it will apply to all components of the continuous design variables vector.

Examples

Two continuous design variables, both scaled by the characteristic value 4.0

```plaintext
continuous_design = 2
initial_point -1.2 1.0
lower_bounds  -200 0.001
upper_bounds  200 2.0
descriptors   'x1' 'x2'
scale_types = 'value' 'none'
scales = 10.0
```

descriptors

- Keywords Area
- variables
- continuous_design
- descriptors

Labels for the variables
6.4. VARIABLES

Specification

Alias: cdv_descriptors
Argument(s): STRINGLIST
Default: cdv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.6 discrete_design_range

- Keywords Area
- variables
- discrete_design_range

Discrete design variables; each defined by an integer interval

Topics

This keyword is related to the topics:

- discrete_variables
- design_variables

Specification

Alias: none
Argument(s): INTEGER
Default: no discrete design variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>lower_bounds</td>
<td></td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Optional</td>
<td>upper_bounds</td>
<td></td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

These variables take on a range of integer values from the specified lower bound to the specified upper bound.
The details of how to specify this discrete variable are located on the discrete_variables page.
initial_point

- Keywords Area
- variables
- discrete_design_range
- initial_point

Initial values

**Specification**

Alias: ddv_initial_point
Argument(s): INTEGERLIST
Default: 0

**Description**

The *initial_point* specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

**lower_bounds**

- Keywords Area
- variables
- discrete_design_range
- lower_bounds

Specify minimum values

**Specification**

Alias: ddv_lower_bounds
Argument(s): INTEGERLIST
Default: INT_MIN

**Description**

Specify minimum values

**upper_bounds**

- Keywords Area
- variables
- discrete_design_range
- upper_bounds

Specify maximum values
6.4. VARIABLES

Specification

Alias: ddv_upper_bounds

Argument(s): INTEGERLIST

Default: INT_MAX

Description

Specify maximum values

descriptors

- Keywords Area
- variables
- discrete_design_range
- descriptors

Labels for the variables

Specification

Alias: ddv_descriptors

Argument(s): STRINGLIST

Default: ddrv_.{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.7 discrete_design_set

- Keywords Area
- variables
- discrete_design_set

Set-valued discrete design variables

Topics

This keyword is related to the topics:

- discrete_variables
- design_variables

Specification

Alias: none

Argument(s): none
## Description

Discrete design variables whose values come from a set of admissible elements. Each variable specified must be of type integer, string, or real.

### integer

- **Keywords Area**
- **variables**
- **discrete_design_set**
- **integer**

Integer-valued discrete design variables

### Topics

This keyword is related to the topics:

- **discrete_variables**
- **design_variables**

### Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** no discrete design set integer variables
6.4. VARIABLES

<table>
<thead>
<tr>
<th>Optional</th>
<th>elements_per_variable</th>
<th>Number of admissible elements for each set variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>elements</td>
<td>The permissible values for each discrete variable</td>
</tr>
<tr>
<td>Optional</td>
<td>categorical</td>
<td>Whether the set-valued variables are categorical or relaxable</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

### Description

A design variable whose values come from a specified set of admissible integers. The details of how to specify this discrete variable are located on the `discrete_variables` page.

### Examples

Four integer variables whose values will be selected from the following sets during the search for an optimal design. $y_1 \in \{0, 1\}$, $y_2 \in \{0, 1\}$, $y_3 \in \{0, 5\}$ and $y_4 \in \{10, 15, 20, 23\}$.

```plaintext
discrete_design_set
integer 4
    descriptors   'y1'  'y2'  'y3'  'y4'
    elements_per_variable  2   2   2   4
    elements       0 1 0 1 0 5 10 15 20 23
```

### elements_per_variable

- **Keywords Area**
- **variables**
- **discrete_design_set**
- **integer**
- **elements_per_variable**

Number of admissible elements for each set variable
CHAPTER 6. KEYWORDS AREA

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

elements

• Keywords Area
• variables
• discrete_design_set
• integer
• elements

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): INTEGERLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete_variables page.

categorical

• Keywords Area
• variables
• discrete_design_set
• integer
• categorical

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none

Argument(s): STRINGLIST
Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```
discrete_design_set
  integer 1
    elements 2 4 7
  descriptor ‘rotor_blades’
  categorical ‘no’
```

adjacency_matrix

- Keywords Area
- variables
- discrete_design_set
- integer
- categorical
- adjacency_matrix

1-0 matrix defining which categorical variable levels are related.

Specification

Alias: none
Argument(s): INTEGERLIST
Description

The adjacency_matrix keyword is associated with discrete_design_set variables that are specified to be categorical. Each such variable is associated with one \( k \times k \) symmetric matrix, where \( k \) is the number of values (or levels) of the variable. Entry \( i,j \) of a matrix should be 1 if level \( i \) and level \( j \) are related by some subjective criteria or if \( i = j \); it should be 0 otherwise. The matrices for all variables of the same type (string, real, or integer) are entered sequentially as a list of integers as shown in the examples below.

Default Behavior

The adjacency_matrix keyword is only relevant for discrete_design_set real and discrete_design_set integer variables if one or more of them have been specified to be categorical. It is always relevant for discrete_design_set string variables. If the user does not define an adjacency matrix, the default is method dependent. Currently, the only method that makes use of the adjacency matrix is mesh_adaptive_search, which uses a tri-diagonal adjacency matrix by default.

Expected Output

The expected output is method dependent.

Usage Tips

If an adjacency matrix is defined for one type of (categorical) discrete_design_set variable, it must be defined for all variables of that type, even for those not defined to be categorical. Those for the non-categorical set variables will be ignored.

Examples

The following example shows a variables specification where some real and some integer discrete_design_set variables are categorical.

```
variables
  continuous_design = 3
    initial_point -1.0 1.5 2.0
    upper_bounds 10.0 10.0 10.0
    descriptors 'x1' 'x2' 'x3'
  discrete_design_range = 2
    initial_point 2 2
    lower_bounds 1 1
    upper_bounds 4 9
    descriptors 'y1' 'y2'
  discrete_design_set
    real = 2
      elements_per_variable = 4 5
      elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
      descriptors 'y3' 'y4'
      categorical 'no' 'yes'
      adjacency_matrix
        1 1 0 0  # Begin entry of 4x4 matrix for y3
        1 1 0
        0 1 1
        0 0 1 1  # Begin entry of 5x5 matrix for y4
      1 0 1 0
      0 1 0 1
    integer = 2
      elements_per_variable = 2 3
      elements = 4 7 8 9 12
      descriptors 'z1' 'z2'
      categorical 'yes' 'yes'
```
Note that for the real case, the user wants to define an adjacency matrix for the categorical variable, so adjacency matrices for both variables must be specified. The matrix for the first one will be ignored. Note that no adjacency matrix is specified for either integer categorical variable. The default will be used in both cases. Currently the only method taking advantage of adjacency matrices is `mesh_adaptive_search`, which uses a tri-diagonal adjacency matrix by default. Thus, the matrices used would be

\[
\begin{align*}
z_1: & \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \\
z_2: & \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}
\end{align*}
\]

The following example shows a variables specification for string variables. Note that string variables are always considered to be categorical. If an adjacency matrix is not specified, a method-dependent default matrix will be used.

```
variables,
  continuous_design = 2
  initial_point 0.5 0.5
  lower_bounds 0. 0.
  upper_bounds 1. 1.
  descriptors = 'x' 'y'
discrete_design_set string = 1
elements = 'aniso1' 'aniso2' 'iso1' 'iso2' 'iso3'
descriptors = 'ancomp'
  adjacency_matrix 1 1 0 0 0
              1 1 0 0 0
              0 0 1 1 1
              0 0 1 1 1
```

See Also

These keywords may also be of interest:

- `mesh_adaptive_search`

initial_point

- Keywords Area
- variables
- discrete_design_set
- integer
- initial_point

Initial values

Specification

Alias: none

Argument(s): INTEGRALIST

Default: middle set value, or rounded down
**Description**

The *initial_point* specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

**descriptors**

- *Keywords Area*
- *variables*
- *discrete_design_set*
- *integer*
- *descriptors*

Labels for the variables

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Description**

The optional variables labels specification *descriptors* is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

**string**

- *Keywords Area*
- *variables*
- *discrete_design_set*
- *string*

String-valued discrete design set variables

**Topics**

This keyword is related to the topics:

- *discrete_variables*
- *design_variables*

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** no discrete design set string variables
### Description

Discrete design variables whose values come from a specified set of admissible strings. The details of how to specify this discrete variable are located on the `discrete_variables` page. Each string element value must be quoted and may contain alphanumeric, dash, underscore, and colon. White space, quote characters, and back-slash/metacharacters are not permitted.

### Examples

Two string variables whose values will be selected from the set of provided elements. The first variable, 'linear solver', takes on values from a set of three possible elements and the second variable, 'mesh_file', from a set of two possible elements.

```plaintext
discrete_design_set
  string 2
descriptors ‘linear_solver’ ‘mesh_file’
elements_per_variable 3 2
  elements ‘cg’ ‘gmres’ ‘direct’
    ‘mesh64.exo’ ‘mesh128.exo’
```

### elements_per_variable

- `Keywords Area`
- `variables`
- `discrete_design_set`
- `string`
- `elements_per_variable`

Number of admissible elements for each set variable.
CHAPTER 6. KEYWORDS AREA

Specification

Alias: num_set_values
  Argument(s): INTEGERLIST
  Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

elements
  • Keywords Area
  • variables
  • discrete_design_set
  • string
  • elements

The permissible values for each discrete variable

Specification

Alias: set_values
  Argument(s): STRINGLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete_variables page.

adjacency_matrix
  • Keywords Area
  • variables
  • discrete_design_set
  • string
  • adjacency_matrix

1-0 matrix defining which categorical variable levels are related.

Specification

Alias: none
  Argument(s): INTEGERLIST
6.4. VARIABLES

Description

The **adjacency_matrix** keyword is associated with **discrete_design_set** variables that are specified to be categorical. Each such variable is associated with one \( k \times k \) symmetric matrix, where \( k \) is the number of values (or levels) of the variable. Entry \( i, j \) of a matrix should be 1 if level \( i \) and level \( j \) are related by some subjective criteria or if \( i = j \); it should be 0 otherwise. The matrices for all variables of the same type (**string**, **real**, or **integer**) are entered sequentially as a list of integers as shown in the examples below.

**Default Behavior**

The **adjacency_matrix** keyword is only relevant for **discrete_design_set** **real** and **discrete_design_set** **integer** variables if one or more of them have been specified to be categorical. It is always relevant for **discrete_design_set** **string** variables. If the user does not define an adjacency matrix, the default is method dependent. Currently, the only method that makes use of the adjacency matrix is **mesh_adaptive_search**, which uses a tri-diagonal adjacency matrix by default.

**Expected Output**

The expected output is method dependent.

**Usage Tips**

If an adjacency matrix is defined for one type of (categorical) **discrete_design_set** variable, if must be defined for all variables of that type, even for those not defined to be categorical. Those for the non-categorical set variables will be ignored.

**Examples**

The following example shows a variables specification where some real and some integer **discrete_design_set** variables are categorical.

```
vARIABLES
  continuous_design = 3
      initial_point  -1.0  1.5  2.0
      lower_bounds  -10.0 -10.0 -10.0
      upper_bounds  10.0  10.0  10.0
      descriptors   'x1' 'x2' 'x3'
  discrete_design_range = 2
      initial_point  2  2
      lower_bounds  1  1
      upper_bounds  4  9
      descriptors   'y1' 'y2'
  discrete_design_set
      real = 2
         elements_per_variable = 4 5
         elements              = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
         descriptors          'y3' 'y4'
         categorical         'no' 'yes'
         adjacency_matrix    1 1 0 0 # Begin entry of 4x4 matrix for y3
                                1 1 1
                                0 1 1
                                0 0 1
                                1 0 1 1 0 # Begin entry of 5x5 matrix for y4
                               0 1 0 1 0
                              1 0 1 0 1
                             0 1 0 1 0
                            1 0 1 0 1
      integer = 2
         elements_per_variable = 2 3
         elements              = 4 7 8 9 12
         descriptors          'z1' 'z2'
         categorical         'yes' 'yes'
```
Note that for the real case, the user wants to define an adjacency matrix for the categorical variable, so adjacency matrices for both variables must be specified. The matrix for the first one will be ignored. Note that no adjacency matrix is specified for either integer categorical variable. The default will be used in both cases. Currently the only method taking advantage of adjacency matrices is `mesh_adaptive_search`, which uses a tri-diagonal adjacency matrix by default. Thus, the matrices used would be

```
z1: 1 1
    1 1
z2: 1 1 0
    1 1 1
    0 1 1
```

The following example shows a variables specification for string variables. Note that string variables are always considered to be categorical. If an adjacency matrix is not specified, a method-dependent default matrix will be used.

```
variables,
    continuous_design = 2
    initial_point 0.5 0.5
    lower_bounds 0. 0.
    upper_bounds 1. 1.
    descriptors = 'x' 'y'
discrete_design_set string = 1
    elements = 'aniso1' 'aniso2' 'iso1' 'iso2' 'iso3'
    descriptors = 'ancomp'
    adjacency_matrix 1 1 0 0 0
    1 1 0 0 0
    0 0 1 1 1
    0 0 1 1 1
    0 0 1 1 1
```

See Also

These keywords may also be of interest:

- `mesh_adaptive_search`

**initial_point**

- Keywords Area
- variables
- discrete_design_set
- string
- initial_point

Initial values

**Specification**

Alias: none

Argument(s): STRINGLIST

Default: middle set value, or rounded down
6.4. VARIABLES

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors
- Keywords Area
- variables
- discrete_design_set
- string
- descriptors
Labels for the variables

Specification
Alias: none
Argument(s): STRINGLIST

Description
The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.
The default descriptor strings use a root string plus a numeric identifier.

real
- Keywords Area
- variables
- discrete_design_set
- real
Real-valued discrete design variables

Topics
This keyword is related to the topics:
- discrete_variables
- design_variables

Specification
Alias: none
Argument(s): INTEGER
Default: no discrete design set real variables
### Description

A design variable whose values come from a specified set of admissible reals. The details of how to specify this discrete variable are located on the `discrete_variables` page.

### Examples

Two continuous, restricted variables whose values will be selected from the following sets during the search for an optimal design. \( y_1 \in \{0.25, 1.25, 2.25, 3.25, 4.25\} \), \( y_2 \in \{0, 5\} \)

```plaintext

discrete_design_set
  real 2
  descriptors 'y1' 'y2'
  elements_per_variable 5 2
  elements 0.25 1.25 2.25 3.25 4.25 0 5
```

### elements_per_variable

- **Keywords Area**
- **variables**
- **discrete_design_set**
- **real**
- **elements_per_variable**

Number of admissible elements for each set variable
6.4. VARIABLES

Specification

Alias: num_set_values
  Argument(s): INTEGERLIST
  Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

elements
  • Keywords Area
  • variables
  • discrete_design_set
  • real
  • elements

The permissible values for each discrete variable

Specification

Alias: set_values
  Argument(s): REALLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete_variables page.

categorical
  • Keywords Area
  • variables
  • discrete_design_set
  • real
  • categorical

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none
  Argument(s): STRINGLIST
### Keywords Area

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>adjacency_matrix</td>
<td>1-0 matrix defining which categorical variable levels are related.</td>
</tr>
</tbody>
</table>

#### Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include ‘yes’, ‘no’, ‘true’, and ‘false’, or any abbreviation in [yYnNtTfF][.]∗

#### Examples

Discrete design set variable, ‘rotor blades’, can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and ‘rotor blades’ can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```plaintext
discrete_design_set
  integer 1
  elements 2 4 7
descriptor 'rotor_blades'
categorical 'no'
```

#### Specification

**Alias:** none  
**Argument(s):** INTEGERLIST
Description

The `adjacency_matrix` keyword is associated with `discrete_design_set` variables that are specified to be categorical. Each such variable is associated with one \( k \times k \) symmetric matrix, where \( k \) is the number of values (or levels) of the variable. Entry \( i,j \) of a matrix should be 1 if level \( i \) and level \( j \) are related by some subjective criteria or if \( i = j \); it should be 0 otherwise. The matrices for all variables of the same type (`string`, `real`, or `integer`) are entered sequentially as a list of integers as shown in the examples below.

Default Behavior

The `adjacency_matrix` keyword is only relevant for `discrete_design_set` `real` and `discrete_design_set` `integer` variables if one or more of them have been specified to be categorical. It is always relevant for `discrete_design_set` `string` variables. If the user does not define an adjacency matrix, the default is method dependent. Currently, the only method that makes use of the adjacency matrix is `mesh_adaptive_search`, which uses a tri-diagonal adjacency matrix by default.

Expected Output

The expected output is method dependent.

Usage Tips

If an adjacency matrix is defined for one type of (categorical) `discrete_design_set` variable, it must be defined for all variables of that type, even for those not defined to be categorical. Those for the non-categorical set variables will be ignored.

Examples

The following example shows a variables specification where some real and some integer `discrete_design_set` variables are categorical.

```
variables
  continuous_design = 3
  initial_point -1.0  1.5  2.0
  lower_bounds  -10.0 -10.0 -10.0
  upper_bounds  10.0  10.0  10.0
  descriptors  'x1'  'x2'  'x3'
  discrete_design_range = 2
  initial_point  2  2
  lower_bounds  1  1
  upper_bounds  4  9
  descriptors  'y1'  'y2'
  discrete_design_set
    real = 2
    elements_per_variable = 4 5
    elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
    descriptors  'y3'  'y4'
    categorical 'no' 'yes'
    adjacency_matrix
      1 1 0 0  # Begin entry of 4x4 matrix for y3
      1 1 1 0
      0 1 1 1
      0 0 1 1
      1 0 1 0 1 # Begin entry of 5x5 matrix for y4
      0 1 0 1 0
      1 0 1 0 1
      0 1 0 1 0
      1 0 1 0 1
    integer = 2
    elements_per_variable = 2 3
    elements = 4 7 8 9 12
    descriptors  'z1' 'z2'
    categorical 'yes' 'yes'
```
Note that for the real case, the user wants to define an adjacency matrix for the categorical variable, so adjacency matrices for both variables must be specified. The matrix for the first one will be ignored. Note that no adjacency matrix is specified for either integer categorical variable. The default will be used in both cases. Currently the only method taking advantage of adjacency matrices is `mesh_adaptive_search`, which uses a tri-diagonal adjacency matrix by default. Thus, the matrices used would be

\[
\begin{array}{cccc}
1 & 1 & & \\
1 & 1 & & \\
0 & 1 & 1 & \\
1 & 1 & 0 & \\
1 & 1 & 1 & \\
\end{array}
\]

The following example shows a variables specification for string variables. Note that string variables are always considered to be categorical. If an adjacency matrix is not specified, a method-dependent default matrix will be used.

```plaintext
variables,
    continuous_design = 2
    initial_point 0.5 0.5
    lower_bounds 0. 0.
    upper_bounds 1. 1.
    descriptors = 'x' 'y'
discrete_design_set string = 1
elements = 'aniso1' 'aniso2' 'iso1' 'iso2' 'iso3'
descriptors = 'ancomp'
adjacency_matrix 1 1 0 0 0
    1 1 0 0 0
    0 0 1 1 1
    0 0 1 1 1
    1 1 0 0 0
```

See Also

These keywords may also be of interest:

- `mesh_adaptive_search`

**initial_point**

- `Keywords Area`
- `variables`
- `discrete_design_set`
- `real`
- `initial_point`

Initial values

**Specification**

Alias: none

Argument(s): REALLIST

Default: middle set value, or rounded down
6.4. VARIABLES

Description

The `initial_point` specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- discrete_design_set
- real
- descriptors

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Description

The optional variables labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.8 normal_uncertain

- Keywords Area
- variables
- normal_uncertain

Aleatory uncertain variable - normal (Gaussian)

Topics

This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification

Alias: none

Argument(s): INTEGER

Default: no normal uncertain variables
### Description

Within the normal uncertain optional group specification, the number of normal uncertain variables, the means, and standard deviations are required specifications, and the distribution lower and upper bounds and variable descriptors are optional specifications. The normal distribution is widely used to model uncertain variables such as population characteristics. It is also used to model the mean of a sample: as the sample size becomes very large, the Central Limit Theorem states that the distribution of the mean becomes approximately normal, regardless of the distribution of the original variables.

The density function for the normal distribution is:

\[ f(x) = \frac{1}{\sqrt{2\pi \sigma_N}} e^{-\frac{1}{2} \left( \frac{x - \mu_N}{\sigma_N} \right)^2} \]

where \( \mu_N \) and \( \sigma_N \) are the mean and standard deviation of the normal distribution, respectively.

Note that if you specify bounds for a normal distribution, the sampling occurs from the underlying distribution with the given mean and standard deviation, but samples are not taken outside the bounds (see "bounded normal" distribution type in [89]). This can result in the mean and the standard deviation of the sample data being different from the mean and standard deviation of the underlying distribution. For example, if you are sampling from a normal distribution with a mean of 5 and a standard deviation of 3, but you specify bounds of 1 and 7, the resulting mean of the samples will be around 4.3 and the resulting standard deviation will be around 1.6. This is because you have bounded the original distribution significantly, and asymmetrically, since 7 is closer to the original mean than 1.

### Theory

When used with design of experiments and multidimensional parameter studies, distribution bounds are inferred. These bounds are \([\mu - 3\sigma, \mu + 3\sigma]\).

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

**means**

- **Keywords Area**
- **variables**
6.4. VARIABLES

- normal_uncertain
- means

First parameter of the distribution

**Specification**

**Alias:** nuv_means

**Argument(s):** REALLIST

**Description**

Means

**std deviations**

- **Keywords Area**
- variables
- normal_uncertain
- std deviations

Second parameter of the distribution

**Specification**

**Alias:** nuv_std deviations

**Argument(s):** REALLIST

**Description**

Standard deviation

**lower bounds**

- **Keywords Area**
- variables
- normal_uncertain
- lower_bounds

Specify minimum values

**Specification**

**Alias:** nuv_lower_bounds

**Argument(s):** REALLIST

**Default:** -infinity
CHAPTER 6. KEYWORDS AREA

Description
Specify minimum values

upper_bounds

• Keywords Area
• variables
• normal_uncertain
• upper_bounds

Specify maximum values

Specification
Alias: nuv_upper_bounds
  Argument(s): REALLIST
  Default: infinity

Description
Specify maximum values

initial_point

• Keywords Area
• variables
• normal_uncertain
• initial_point

Initial values

Specification
Alias: none
  Argument(s): REALLIST

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).
6.4. VARIABLES

• Keywords Area
• variables
• normal_uncertain
• descriptors

Labels for the variables

Specification

Alias: nuv_descriptors
   Argument(s): STRINGLIST
   Default: nuv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.
   The default descriptor strings use a root string plus a numeric identifier.

6.4.9  lognormal_uncertain

• Keywords Area
• variables
• lognormal_uncertain

Aleatory uncertain variable - lognormal

Topics

This keyword is related to the topics:

• continuous_variables
• aleatory_uncertain_variables

Specification

Alias: none
   Argument(s): INTEGER
   Default: no lognormal uncertain variables

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Description

If the logarithm of an uncertain variable $X$ has a normal distribution, that is $\log X \sim N(\mu, \sigma)$, then $X$ is distributed with a lognormal distribution. The lognormal is often used to model:

1. time to perform some task
2. variables which are the product of a large number of other quantities, by the Central Limit Theorem
3. quantities which cannot have negative values.

Within the lognormal uncertain optional group specification, the number of lognormal uncertain variables, the means, and either standard deviations or error factors must be specified, and the distribution lower and upper bounds and variable descriptors are optional specifications. These distribution bounds can be used to truncate the tails of lognormal distributions, which as for bounded normal, can result in the mean and the standard deviation of the sample data being different from the mean and standard deviation of the underlying distribution (see "bounded lognormal" and "bounded lognormal-n" distribution types in[89]).

For the lognormal variables, one may specify either the mean $\mu$ and standard deviation $\sigma$ of the actual lognormal distribution (option 1), the mean $\mu$ and error factor $\epsilon$ of the actual lognormal distribution (option 2), or the mean $\lambda$ ("lambda") and standard deviation $\zeta$ ("zeta") of the underlying normal distribution (option 3).

The conversion equations from lognormal mean $\mu$ and either lognormal error factor $\epsilon$ or lognormal standard deviation $\sigma$ to the mean $\lambda$ and standard deviation $\zeta$ of the underlying normal distribution are as follows:

$$\zeta = \frac{\ln(\epsilon)}{1.645}$$
$$\zeta^2 = \ln\left(\frac{\sigma^2}{\mu^2} + 1\right)$$
$$\lambda = \ln(\mu) - \frac{\zeta^2}{2}$$

Conversions from $\lambda$ and $\zeta$ back to $\mu$ and $\epsilon$ or $\sigma$ are as follows:

$$\mu = e^{\lambda + \frac{\zeta^2}{2}}$$
\[ \sigma^2 = e^{2\lambda+\zeta^2}(e^{\zeta^2}-1) \]
\[ \epsilon = e^{1.645\zeta} \]

The density function for the lognormal distribution is:
\[ f(x) = \frac{1}{\sqrt{2\pi}\zeta x} e^{-\frac{1}{2}\left(\frac{\ln x - \lambda}{\zeta}\right)^2} \]

**Theory**

When used with design of experiments and multidimensional parameter studies, distribution bounds are inferred. These bounds are \([0, \mu + 3\sigma]\).

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

**lambdas**
- Keywords Area
- variables
- lognormal_uncertain
- lambdas

First parameter of the lognormal distribution (option 3)

**Specification**

**Alias:** lnuv_lambdas

**Argument(s):** REALLIST

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>zetas</td>
<td></td>
<td>Second parameter of the lognormal distribution (option 3)</td>
</tr>
</tbody>
</table>

**Description**

For the lognormal variables, one may specify the mean \(\lambda\) ("lambda") and standard deviation \(\zeta\) ("zeta") of the underlying normal distribution.

**zetas**
- Keywords Area
- variables
- lognormal_uncertain
- lambdas
- zetas

Second parameter of the lognormal distribution (option 3)
Specification

Alias: lnuv_zetas

Argument(s): REALLIST

Description

For the lognormal variables, one may specify the mean $\lambda$ ("lambda") and standard deviation $\zeta$ ("zeta") of the underlying normal distribution.

First parameter of the lognormal distribution (options 1 & 2)

Specification

Alias: lnuv_means

Argument(s): REALLIST

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>std_deviations</td>
<td>Second parameter of the lognormal distribution (option 1)</td>
</tr>
<tr>
<td></td>
<td>error_factors</td>
<td></td>
<td>Second parameter of the lognormal distribution (option 2)</td>
</tr>
</tbody>
</table>

Description

For the lognormal variables, one may specify either the mean $\mu$ and standard deviation $\sigma$ of the actual lognormal distribution, the mean $\mu$ and error factor $\epsilon$ of the actual lognormal distribution.

This corresponds to the mean of the lognormal random variable

std_deviations

- Keywords Area
- variables
- lognormal_uncertain
- means
- std_deviations

Second parameter of the lognormal distribution (option 1)
6.4. VARIABLES

**Specification**

*Alias:* lnuv_std deviations

*Argument(s):* REALLIST

**Description**

For the lognormal variables, one may specify either the mean $\mu$ and standard deviation $\sigma$ of the actual lognormal distribution.

This corresponds to the standard deviation of the lognormal random variable.

**error_factors**

- *Keywords Area*
- *variables*
- *lognormal_uncertain*
- *means*
- *error_factors*

Second parameter of the lognormal distribution (option 2)

**Specification**

*Alias:* lnuv_error_factors

*Argument(s):* REALLIST

**Description**

For the lognormal variables, one may specify the mean $\mu$ and error factor $\epsilon$ of the actual lognormal distribution.

This specifies the error function of the lognormal random variable.

**lower_bounds**

- *Keywords Area*
- *variables*
- *lognormal_uncertain*
- *lower_bounds*

Specify minimum values

**Specification**

*Alias:* lnuv_lower_bounds

*Argument(s):* REALLIST

*Default:* 0
CHAPTER 6. KEYWORDS AREA

Description

Specify minimum values

upper_bounds

- Keywords Area
- variables
- lognormal_uncertain
- upper_bounds

Specify maximum values

Specification

Alias: lnuv_upper_bounds

Argument(s): REALLIST

Default: infinity

Description

Specify maximum values

initial_point

- Keywords Area
- variables
- lognormal_uncertain
- initial_point

Initial values

Specification

Alias: none

Argument(s): REALLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).
6.4. VARIABLES

descriptors
- Keywords Area
- variables
- lognormal_uncertain
- descriptors

Labels for the variables

Specification

Alias: lnuv_descriptors
Argument(s): STRINGLIST
Default: lnuv_\{i\}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.10 uniform_uncertain

- Keywords Area
- variables
- uniform_uncertain

Aleatory uncertain variable - uniform

Topics

This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification

Alias: none
Argument(s): INTEGER
Default: no uniform uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Description**

Within the uniform uncertain optional group specification, the number of uniform uncertain variables and the distribution lower and upper bounds are required specifications, and variable descriptors is an optional specification. The uniform distribution has the density function:

\[ f(x) = \frac{1}{U_U - L_U} \]

where \( U_U \) and \( L_U \) are the upper and lower bounds of the uniform distribution, respectively. The mean of the uniform distribution is \( \frac{U_U + L_U}{2} \) and the variance is \( \frac{(U_U - L_U)^2}{12} \).

**Theory**

Note that this distribution is a special case of the more general beta distribution.

**lower_bounds**

- **Keywords Area**
- **variables**
- **uniform_uncertain**
- **lower_bounds**

Specify minimum values

**Specification**

Alias: uuv_lower_bounds

Argument(s): REALLIST

**Description**

Specify minimum values
6.4. VARIABLES

upper_bounds

- Keywords Area
- variables
- uniform_uncertain
- upper_bounds

Specify maximum values

Specification

Alias: uuv_upper_bounds
Argument(s): REALLIST

Description

Specify maximum values

initial_point

- Keywords Area
- variables
- uniform_uncertain
- initial_point

Initial values

Specification

Alias: none
Argument(s): REALLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- uniform_uncertain
- descriptors

Labels for the variables
CHAPTER 6. KEYWORDS AREA

Specification

Alias: uuv_descriptors
Argument(s): STRINGLIST
Default: uuv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.11 loguniform_uncertain

- Keywords Area
- variables
- loguniform_uncertain

Aleatory uncertain variable - loguniform

Topics

This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification

Alias: none
Argument(s): INTEGER
Default: no loguniform uncertain variables

<p>| Required/-| Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>lower_bounds</td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>upper_bounds</td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>initial_point</td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>descriptors</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>
6.4. VARIABLES

Description
If the logarithm of an uncertain variable X has a uniform distribution, that is $\log X \sim U(L_{LU}, U_{LU})$, then X is distributed with a loguniform distribution. The distribution lower bound is $L_{LU}$ and upper bound is $U_{LU}$. The loguniform distribution has the density function:

$$f(x) = \frac{1}{x(lnU_{LU} - lnL_{LU})}$$

Theory
For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

lower_bounds
- Keywords Area
- variables
- loguniform_uncertain
- lower_bounds

Specify minimum values

Specification
Alias: luuv_lower_bounds
Argument(s): REALLIST

Description
Specify minimum values

upper_bounds
- Keywords Area
- variables
- loguniform_uncertain
- upper_bounds

Specify maximum values

Specification
Alias: luuv_upper_bounds
Argument(s): REALLIST
**Description**

Specify maximum values

**initial_point**

- Keywords Area
- variables
- loguniform_uncertain
- initial_point

Initial values

**Specification**

Alias: none

*Argument(s):* REALLIST

**Description**

The *initial_point* specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

**descriptors**

- Keywords Area
- variables
- loguniform_uncertain
- descriptors

Labels for the variables

**Specification**

Alias: luuv_descriptors

*Argument(s):* STRINGLIST

*Default:* luuv_{i}

**Description**

The optional variables labels specification *descriptors* is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.
6.4. VARIABLES

6.4.12 triangular_uncertain

Keywords Area
  • variables
  • triangular_uncertain

Aleatory uncertain variable - triangular

Topics

This keyword is related to the topics:
  • continuous_variables
  • aleatory_uncertain_variables

Specification

Alias: none
Argument(s): INTEGER
Default: no triangular uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>modes</td>
<td>Distribution parameter</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>lower_bounds</td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>upper_bounds</td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>initial_point</td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

The triangular distribution is often used when one does not have much data or information, but does have an estimate of the most likely value and the lower and upper bounds. Within the triangular uncertain optional group specification, the number of triangular uncertain variables, the modes, and the distribution lower and upper bounds are required specifications, and variable descriptors is an optional specification.

The density function for the triangular distribution is:

\[ f(x) = \frac{2(x - L_T)}{(U_T - L_T)(M_T - L_T)} \]

if \( L_T \leq x \leq M_T \), and

\[ f(x) = \frac{2(U_T - x)}{(U_T - L_T)(U_T - M_T)} \]

if \( M_T \leq x \leq U_T \), and 0 elsewhere. In these equations, \( L_T \) is the lower bound, \( U_T \) is the upper bound, and \( M_T \) is the mode of the triangular distribution.
modes
  • Keywords Area
  • variables
  • triangular_uncertain
  • modes

Distribution parameter

Specification
Alias: tvu_modes
Argument(s): REALLIST

Description
Specify the modes

lower_bounds
  • Keywords Area
  • variables
  • triangular_uncertain
  • lower_bounds

Specify minimum values

Specification
Alias: tvu_lower_bounds
Argument(s): REALLIST

Description
Specify minimum values

upper_bounds
  • Keywords Area
  • variables
  • triangular_uncertain
  • upper_bounds

Specify maximum values
6.4. VARIABLES

Specification

Alias: tuv_upper_bounds
Argument(s): REALLIST

Description

Specify maximum values

initial_point

- Keywords Area
- variables
- triangular_uncertain
- initial_point

Initial values

Specification

Alias: none
Argument(s): REALLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- triangular_uncertain
- descriptors

Labels for the variables

Specification

Alias: tuv_descriptors
Argument(s): STRINGLIST
Default: tuv_{i}
Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.13 exponential_uncertain

- Keywords Area
- variables
- exponential_uncertain

Aleatory uncertain variable - exponential

Topics

This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification

Alias: none

Argument(s): INTEGER

Default: no exponential uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>betas</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>initial_point</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>descriptors</td>
<td></td>
</tr>
</tbody>
</table>

Description

The exponential distribution is often used for modeling failure rates.

The density function for the exponential distribution is given by:

\[ f(x) = \frac{1}{\beta} e^{-\frac{x}{\beta}} \]

where \( \mu_E = \beta \) and \( \sigma^2_E = \beta^2 \).

Note that this distribution is a special case of the more general gamma distribution.
6.4. VARIABLES

Theory
When used with design of experiments and multidimensional parameter studies, distribution bounds are inferred. These bounds are $[0, \mu + 3\sigma]$.

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

betas
- Keywords Area
- variables
- exponential_uncertain
- betas

Parameter of the exponential distribution

Specification
Alias: euv_betas
Argument(s): REALLIST

Description
Specifies the list of $\beta$ parameters to define the distributions of the exponential random variables. Length must match the other parameters and the number of exponential random variables.

initial_point
- Keywords Area
- variables
- exponential_uncertain
- initial_point

Initial values

Specification
Alias: none
Argument(s): REALLIST

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).
descriptors

- Keywords Area
- variables
- exponential_uncertain
- descriptors

Labels for the variables

**Specification**

**Alias:** euv_descriptors

**Argument(s):** STRINGLIST

**Default:** euv_{i}

**Description**

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

### 6.4.14 beta_uncertain

- Keywords Area
- variables
- beta_uncertain

Aleatory uncertain variable - beta

**Topics**

This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** no beta uncertain variables

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional</th>
<th>Group</th>
<th></th>
<th>Description</th>
</tr>
</thead>
</table>
6.4. VARIABLES

<table>
<thead>
<tr>
<th>Required</th>
<th>alphas</th>
<th>First parameter of the beta distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>betas</td>
<td>Second parameter of the beta distribution</td>
</tr>
<tr>
<td>Required</td>
<td>lower_bounds</td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Required</td>
<td>upper_bounds</td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

Within the beta uncertain optional group specification, the number of beta uncertain variables, the alpha and beta parameters, and the distribution upper and lower bounds are required specifications, and the variable descriptors is an optional specification. The beta distribution can be helpful when the actual distribution of an uncertain variable is unknown, but the user has a good idea of the bounds, the mean, and the standard deviation of the uncertain variable. The density function for the beta distribution is

\[
f(x) = \frac{\Gamma(\alpha + \beta) (x - L_B)^{\alpha - 1} (U_B - x)^{\beta - 1}}{\Gamma(\alpha) \Gamma(\beta)} \frac{1}{(U_B - L_B)^{\alpha + \beta - 1}}
\]

where \( \Gamma(\alpha) \) is the gamma function and \( B(\alpha, \beta) = \frac{\Gamma(\alpha) \Gamma(\beta)}{\Gamma(\alpha + \beta)} \) is the beta function. To calculate the mean and standard deviation from the alpha, beta, upper bound, and lower bound parameters of the beta distribution, the following expressions may be used.

\[
\mu_B = L_B + \frac{\alpha}{\alpha + \beta} (U_B - L_B)
\]

\[
\sigma_B^2 = \frac{\alpha \beta}{(\alpha + \beta)^2 (\alpha + \beta + 1)} (U_B - L_B)^2
\]

Solving these for \( \alpha \) and \( \beta \) gives:

\[
\alpha = \frac{(\mu_B - L_B)(\mu_B - L_B)(U_B - \mu_B) - \sigma_B^2}{\sigma_B^2 (U_B - L_B)}
\]

\[
\beta = \frac{(U_B - \mu_B)(\mu_B - L_B)(U_B - \mu_B) - \sigma_B^2}{\sigma_B^2 (U_B - L_B)}
\]

Note that the uniform distribution is a special case of this distribution for parameters \( \alpha = \beta = 1 \).

Theory

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.
**alphas**
- Keywords Area
- variables
- beta_uncertain
- alphas

First parameter of the beta distribution

**Specification**
**Alias:** buv_alphas
**Argument(s):** REALLIST

**Description**
Specifies the list of $\alpha$ parameters to define the distributions of the beta random variables. Length must match the other parameters and the number of beta random variables.

**betas**
- Keywords Area
- variables
- beta_uncertain
- betas

Second parameter of the beta distribution

**Specification**
**Alias:** buv_betas
**Argument(s):** REALLIST

**Description**
Specifies the list of $\beta$ parameters to define the distributions of the beta random variables. Length must match the other parameters and the number of beta random variables.

**lower_bound**
- Keywords Area
- variables
- beta_uncertain
- lower_bound

Specify minimum values
6.4. VARIABLES

Specification

Alias: buv_lower_bounds
Argument(s): REALLIST

Description
Specify minimum values

upper_bounds
- Keywords Area
- variables
- beta_uncertain
- upper_bounds

Specify maximum values

Specification

Alias: buv_upper_bounds
Argument(s): REALLIST

Description
Specify maximum values

initial_point
- Keywords Area
- variables
- beta_uncertain
- initial_point

Initial values

Specification

Alias: none
Argument(s): REALLIST

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).
CHAPTER 6. KEYWORDS AREA

descriptors

- Keywords Area
- variables
- beta_uncertain
- descriptors

Labels for the variables

Specification

Alias: buv_descriptors

Argument(s): STRINGLIST

Default: buv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.15 gamma_uncertain

- Keywords Area
- variables
- gamma_uncertain

Aleatory uncertain variable - gamma

Topics

This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification

Alias: none

Argument(s): INTEGER

Default: no gamma uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
</table>


### 6.4. VARIABLES

<table>
<thead>
<tr>
<th></th>
<th>Required</th>
<th>Required</th>
<th>Optional</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>alphas</td>
<td>betas</td>
<td>initial_point</td>
<td>descriptors</td>
</tr>
<tr>
<td></td>
<td>First parameter of the gamma distribution</td>
<td>Second parameter of the gamma distribution</td>
<td>Initial values</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

**Description**

The gamma distribution is sometimes used to model time to complete a task, such as a repair or service task. It is a very flexible distribution with its shape governed by alpha and beta.

The density function for the gamma distribution is given by:

\[
f(x) = \frac{x^{\alpha-1}e^{-x/\beta}}{\beta^\alpha\Gamma(\alpha)}
\]

where \(\mu_{GA} = \alpha\beta\) and \(\sigma_{GA}^2 = \alpha\beta^2\). Note that the exponential distribution is a special case of this distribution for parameter \(\alpha = 1\).

**Theory**

When used with design of experiments and multidimensional parameter studies, distribution bounds are inferred. These bounds are \([0, \mu + 3\sigma]\).

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

**alphas**

- **Keywords** Area
- **variables**
- **gamma_uncertain**
- **alphas**

First parameter of the gamma distribution

**Specification**

Alias: gauv_alphas

Argument(s): REALLIST

**Description**

Specifies the list of \(\alpha\) parameters to define the distributions of the gamma random variables. Length must match the other parameters and the number of gamma random variables.
betas

- Keywords Area
- variables
- gamma_uncertain
- betas

Second parameter of the gamma distribution

**Specification**

**Alias:** gauv.betas

**Argument(s):** REALLIST

**Description**

Specifies the list of $\beta$ parameters to define the distributions of the gamma random variables. Length must match the other parameters and the number of gamma random variables.

initial_point

- Keywords Area
- variables
- gamma_uncertain
- initial_point

Initial values

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Description**

The `initial_point` specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- gamma_uncertain
- descriptors

Labels for the variables
6.4. VARIABLES

Specification

Alias: gauv_descriptors
  Argument(s): STRINGLIST
  Default: gauv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.16 gumbel_uncertain

- Keywords Area
- variables
- gumbel_uncertain

Aleatory uncertain variable - gumbel

Topics

This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification

Alias: none
  Argument(s): INTEGER
  Default: no gumbel uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>alphas</td>
<td>betas</td>
<td>First parameter of the gumbel distribution</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td>descriptors</td>
<td>Second parameter of the gumbel distribution</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>
Description

The Gumbel distribution is also referred to as the Type I Largest Extreme Value distribution. The distribution of maxima in sample sets from a population with a normal distribution will asymptotically converge to this distribution. It is commonly used to model demand variables such as wind loads and flood levels.

The density function for the Gumbel distribution is given by:
\[
f(x) = \alpha e^{-\alpha(x-\beta)} \exp(-e^{-\alpha(x-\beta)})
\]

where \( \mu_{GU} = \beta + \frac{0.5772}{\alpha} \) and \( \sigma_{GU} = \frac{\pi}{\sqrt{6\alpha}} \).

Theory

When used with design of experiments and multidimensional parameter studies, distribution bounds are inferred. These bounds are \([\mu - 3\sigma, \mu + 3\sigma]\).

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

alphas

- **Keywords Area**
- **variables**
- **gumbel_uncertain**
- **alphas**

First parameter of the gumbel distribution

Specification

**Alias:** guuv_alphas

**Argument(s):** REALLIST

Description

Specifies the list of \( \beta \) parameters to define the distributions of the gumbel random variables. Length must match the other parameters and the number of gumbel random variables.

betas

- **Keywords Area**
- **variables**
- **gumbel_uncertain**
- **betas**

Second parameter of the gumbel distribution
6.4. VARIABLES

**Specification**

**Alias:** guuv\_betas  
**Argument(s):** REALLIST

**Description**

Specifies the list of $\beta$ parameters to define the distributions of the gumbel random variables. Length must match the other parameters and the number of gumbel random variables.

**initial\_point**

- **Keywords Area**
- **variables**
- **gumbel\_uncertain**
- **initial\_point**

Initial values

**Specification**

**Alias:** none  
**Argument(s):** REALLIST

**Description**

The *initial\_point* specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

**descriptors**

- **Keywords Area**
- **variables**
- **gumbel\_uncertain**
- **descriptors**

Labels for the variables

**Specification**

**Alias:** guuv\_descriptors  
**Argument(s):** STRINGLIST  
**Default:** guuv\_\{i\}
CHAPTER 6. KEYWORDS AREA

Description
The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.17 frechet_uncertain

- Keywords Area
- variables
- frechet_uncertain

Aleatory uncertain variable - Frechet

Topics
This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification
Alias: none

Argument(s): INTEGER

Default: no frechet uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>alphas</td>
<td></td>
<td>First parameter of the Frechet distribution</td>
</tr>
<tr>
<td>Required</td>
<td>betas</td>
<td></td>
<td>Second parameter of the Frechet distribution</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description
The Frechet distribution is also referred to as the Type II Largest Extreme Value distribution. The distribution of maxima in sample sets from a population with a lognormal distribution will asymptotically converge to this distribution. It is commonly used to model non-negative demand variables.

The density function for the frechet distribution is:

\[
f(x) = \frac{\alpha}{\beta} \left(\frac{\beta}{x}\right)^{\alpha+1} e^{-\frac{\beta}{x}}
\]

where \( \mu_F = \beta \Gamma(1 - \frac{1}{\alpha}) \) and \( \sigma_F^2 = \beta^2 \left[ \Gamma(1 - \frac{2}{\alpha}) - \Gamma^2(1 - \frac{1}{\alpha}) \right] \)
Theory

When used with design of experiments and multidimensional parameter studies, distribution bounds are inferred. These bounds are \([0, \mu + 3\sigma]\).

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

alphas

- Keywords Area
- variables
- frechet_uncertain
- alphas

First parameter of the Frechet distribution

Specification

Alias: fuv_alphas

Argument(s): REALLIST

Description

Specifies the list of \(\alpha\) parameters to define the distributions of the Frechet random variables. Length must match the other parameters and the number of Frechet random variables.

betas

- Keywords Area
- variables
- frechet_uncertain
- betas

Second parameter of the Frechet distribution

Specification

Alias: fuv_betas

Argument(s): REALLIST

Description

Specifies the list of \(\beta\) parameters to define the distributions of the Frechet random variables. Length must match the other parameters and the number of Frechet random variables.
initial_point

- Keywords Area
- variables
- frechet_uncertain
- initial_point

Initial values

Specification

Alias: none
Argument(s): REALLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- frechet_uncertain
- descriptors

Labels for the variables

Specification

Alias: fuv_descriptors
Argument(s): STRINGLIST
Default: fuv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.18 weibull_uncertain

- Keywords Area
- variables
- weibull_uncertain

Aleatory uncertain variable - Weibull
6.4. VARIABLES

Topics
This keyword is related to the topics:

- continuous_variables
- aleatory_uncertain_variables

Specification

Alias: none

Argument(s): INTEGER

Default: no weibull uncertain variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>alphas</td>
<td>First parameter of the Weibull distribution</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>betas</td>
<td>Second parameter of the Weibull distribution</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

The Weibull distribution is also referred to as the Type III Smallest Extreme Value distribution. The Weibull distribution is commonly used in reliability studies to predict the lifetime of a device. It is also used to model capacity variables such as material strength.

The density function for the Weibull distribution is given by:

\[ f(x) = \frac{\alpha}{\beta} \left(\frac{x}{\beta}\right)^{\alpha-1} e^{-\left(\frac{x}{\beta}\right)^{\alpha}} \]

where \( \mu_W = \beta \Gamma(1 + \frac{1}{\alpha}) \) and \( \sigma_W = \sqrt{\frac{\Gamma(1 + \frac{2}{\alpha})}{\Gamma^2(1 + \frac{1}{\alpha})}} - 1\mu_W \)

alphas

- Keywords Area
- variables
- weibull_uncertain
- alphas

First parameter of the Weibull distribution
CHAPTER 6. KEYWORDS AREA

Specification
Alias: wuv_alphas
   Argument(s): REALLIST

Description
Specifies the list of $\alpha$ parameters to define the distributions of the Weibull random variables. Length must match the other parameters and the number of Weibull random variables.

betas
- Keywords Area
- variables
- weibull_uncertain
- betas

Second parameter of the Weibull distribution

Specification
Alias: wuv_betas
   Argument(s): REALLIST

Description
Specifies the list of $\beta$ parameters to define the distributions of the Weibull random variables. Length must match the other parameters and the number of Weibull random variables.

initial_point
- Keywords Area
- variables
- weibull_uncertain
- initial_point

Initial values

Specification
Alias: none
   Argument(s): REALLIST

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).
6.4. VARIABLES

descriptors
  • Keywords Area
  • variables
  • weibull_uncertain
  • descriptors

Labels for the variables

Specification
Alias: wuv_descriptors
  Argument(s): STRINGLIST
  Default: wuv_{i}

Description
The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.
  The default descriptor strings use a root string plus a numeric identifier.

6.4.19 histogram_bin_uncertain
  • Keywords Area
  • variables
  • histogram_bin_uncertain

Aleatory uncertain variable - continuous histogram

Topics
This keyword is related to the topics:
  • continuous_variables
  • aleatory_uncertain_variables

Specification
Alias: none
  Argument(s): INTEGER
  Default: no histogram bin uncertain variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Optional
pairs_per_variable
Number of pairs defining each histogram bin variable

Required
abscissas
Real abscissas for a bin histogram

Required (Choose One)
ordinates
Ordinates specifying a “skyline” probability density function

Optional
counts
Frequency or relative probability of each bin

Optional
initial_point
Initial values

Optional
descriptors
Labels for the variables

Description

Histogram uncertain variables are typically used to model a set of empirical data. The bin histogram (contrast: histogram_point_uncertain) is a continuous aleatory distribution characterized by bins of non-zero width where the uncertain variable may lie, together with the relative frequencies of each bin. Hence it can be used to specify a marginal probability density function arising from data.

The histogram_bin_uncertain keyword specifies the number of variables to be characterized as continuous histograms. The required sub-keywords are: abscissas (ranges of values the variable can take on) and either ordinates or counts (characterizing each variable’s frequency information). When using histogram bin variables, each variable must be defined by at least one bin (with two bounding value pairs). When more than one histogram bin variable is active, pairs_per_variable can be used to specify unequal apportionment of provided bin pairs among the variables.

The abscissas specification defines abscissa values (“x” coordinates) for the probability density function of each histogram variable. When paired with counts, the specifications provide sets of (x,c) pairs for each histogram variable where c defines a count (i.e., a frequency or relative probability) associated with a bin. If using bins of unequal width and specification of probability densities is more natural, then the counts specification can be replaced with an ordinates specification (“y” coordinates) in order to support interpretation of the input as (x,y) pairs defining the profile of a "skyline” probability density function.

Conversion between the two specifications is straightforward: a count/frequency is a cumulative probability quantity defined from the product of the ordinate density value and the x bin width. Thus, in the cases of bins of equal width, ordinate and count specifications are equivalent. In addition, ordinates and counts may be relative values; it is not necessary to scale them as all user inputs will be normalized.

To fully specify a bin-based histogram with n bins (potentially of unequal width), n+1 (x,c) or (x,y) pairs must be specified with the following features:

- x is the parameter value for the left boundary of a histogram bin and c is the corresponding count for that
bin. Alternatively, \( y \) defines the ordinate density value for this bin within a skyline probability density function. The right boundary of the bin is defined by the left boundary of the next pair.

- the final pair specifies the right end of the last bin and must have a \( c \) or \( y \) value of zero.
- the \( x \) values must be strictly increasing.
- all \( c \) or \( y \) values must be positive, except for the last which must be zero.
- a minimum of two pairs must be specified for each bin-based histogram variable.

### Examples

The `pairs_per_variable` specification provides for the proper association of multiple sets of \((x,c)\) or \((x,y)\) pairs with individual histogram variables. For example, in this input snippet

```plaintext
histogram_bin_uncertain = 2
pairs_per_variable = 3 4
abscissas = 5 8 10 .1 .2 .3 .4
counts = 17 21 0 12 24 12 0
descriptors = 'hbu_1' 'hbu_2'
```

`pairs_per_variable` associates the first 3 \((x,c)\) pairs from `abscissas` and `counts` \{(5,17),(8,21),(10,0)\} with one bin-based histogram variable, where one bin is defined between 5 and 8 with a count of 17 and another bin is defined between 8 and 10 with a count of 21. The following set of 4 \((x,c)\) pairs \{(0.1,12),(0.2,24),(0.3,12),(0.4,0)\} defines a second bin-based histogram variable containing three equal-width bins with counts 12, 24, and 12 (middle bin is twice as probable as the other two).

### See Also

These keywords may also be of interest:

- `histogram_point_uncertain`

### FAQ

**Difference between bin and point histograms:** A (continuous) bin histogram specifies bins of non-zero width, whereas a (discrete) point histogram specifies individual point values, which can be thought of as bins with zero width. In the terminology of LHS[89], the bin pairs specification defines a "continuous linear" distribution and the point pairs specification defines a "discrete histogram" distribution (although the points are real-valued, the number of possible values is finite).

`pairs_per_variable`

- **Keywords Area**
- **variables**
- **histogram_bin_uncertain**
- **pairs_per_variable**

Number of pairs defining each histogram bin variable
**Specification**

**Alias:** num_pairs

**Argument(s):** INTEGERLIST

**Default:** equal distribution

**Description**

By default, the list of abscissas and counts or ordinates will be evenly divided among the histogram_bin_uncertain variables. pairs_per_variable is a list of integers that specify the number of pairs to apportion to each variable.

**abscissas**

- Keywords Area
- variables
- histogram_bin_uncertain
- abscissas

Real abscissas for a bin histogram

**Specification**

**Alias:** huv_bin_abscissas

**Argument(s):** REALLIST

**Description**

A list of real abscissa ("x" coordinate) values characterizing the probability density function for each of the histogram_bin_uncertain variables. These are paired with either counts or ordinates. See histogram_bin_uncertain for details and examples.

**ordinates**

- Keywords Area
- variables
- histogram_bin_uncertain
- ordinates

Ordinates specifying a "skyline" probability density function

**Specification**

**Alias:** huv_bin_ordinates

**Argument(s):** REALLIST
6.4. VARIABLES

Description
The *ordinates* list of real values defines the profile of a "skyline" probability density function by pairing with the specified *abscissas*. See *histogram_bin_uncertain* for details.

**counts**
- **Keywords Area**
- **variables**
- **histogram_bin_uncertain**
- **counts**

Frequency or relative probability of each bin

Specification
**Alias:** huv_bin_counts
**Argument(s):** REALLIST

**Description**
The *counts* list of real values gives the frequency or relative probability for each bin in a *histogram_bin_uncertain* specification. These are paired with the specified *abscissas*. See *histogram_bin_uncertain* for details.

**initial_point**
- **Keywords Area**
- **variables**
- **histogram_bin_uncertain**
- **initial_point**

Initial values

Specification
**Alias:** none
**Argument(s):** REALLIST

**Description**
The *initial_point* specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).


**descriptors**

- **Keywords Area**
- **variables**
- **histogram_bin_uncertain**
- **descriptors**

Labels for the variables

**Specification**

**Alias:** huv_bin_descriptors

**Argument(s):** STRINGLIST

**Default:** hbu.{i}

**Description**

The optional variables labels specification **descriptors** is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

**6.4.20 poisson_uncertain**

- **Keywords Area**
- **variables**
- **poisson_uncertain**

Aleatory uncertain discrete variable - Poisson

**Topics**

This keyword is related to the topics:

- **discrete_variables**
- **aleatory_uncertain_variables**

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** no poisson uncertain variables
**Description**

The Poisson distribution is used to predict the number of discrete events that happen in a single time interval. The random events occur uniformly and independently. The expected number of occurrences in a single time interval is $\lambda$, which must be a positive real number. For example, if events occur on average 4 times per year and we are interested in the distribution of events over six months, $\lambda$ would be 2. However, if we were interested in the distribution of events occurring over 5 years, $\lambda$ would be 20.

The density function for the poisson distribution is given by:

$$f(x) = \frac{\lambda e^{-\lambda}}{x!}$$

where

- $\lambda$ is the expected number of events occurring in a single time interval
- $x$ is the number of events that occur in this time period
- $f(x)$ is the probability that $x$ events occur in this time period

**Theory**

When used with design of experiments and multidimensional parameter studies, distribution bounds are inferred. These bounds are $[0, \mu + 3\sigma]$.

For vector and centered parameter studies, an inferred initial starting point is needed for the uncertain variables. These variables are initialized to their means for these studies.

**lambdas**

- **Keywords** Area
- **variables**
- **poisson_uncertain**
- **lambdas**

The parameter for the Poisson distribution, the expected number of events in the time interval of interest

**Specification**

**Alias:** none

**Argument(s):** REALLIST
Description

The density function for the poisson distribution is given by:

\[ f(x) = \frac{\lambda e^{-\lambda}}{x!} \]

where \( \lambda \) is the frequency of events happening, and \( x \) is the number of events that occur.

initial_point

- Keywords Area
- variables
- poisson_uncertain
- initial_point

Initial values

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- poisson_uncertain
- descriptors

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: puv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.
6.4. VARIABLES

6.4.21 binomial_uncertain

- Keywords Area
- variables
- binomial_uncertain

Aleatory uncertain discrete variable - binomial

Topics

This keyword is related to the topics:

- discrete_variables
- aleatory_uncertain_variables

Specification

Alias: none
Argument(s): INTEGER
Default: no binomial uncertain variables

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>probability_per-trial</td>
<td>Description</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>num_trials</td>
<td>A distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initial_point</td>
<td>parameter</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>descriptors</td>
<td>binomial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initial values</td>
</tr>
</tbody>
</table>

Description

The binomial distribution describes probabilities associated with a series of independent Bernoulli trials. A Bernoulli trial is an event with two mutually exclusive outcomes, such as 0 or 1, yes or no, success or fail. The probability of success remains the same (the trials are independent).

The density function for the binomial distribution is given by:

\[ f(x) = \binom{n}{x} p^x (1 - p)^{n-x} \]

where \( p \) is the probability of failure per trial, \( n \) is the number of trials and \( x \) is the number of successes.

Theory

The binomial distribution is typically used to predict the number of failures or defective items in a total of \( n \) independent tests or trials, where each trial has the probability \( p \) of failing or being defective.
**probability_per_trial**

- **Keywords Area**
- **variables**
- **binomial_uncertain**
- **probability_per_trial**

A distribution parameter for the binomial distribution

**Specification**

**Alias:** prob_per_trial

**Argument(s):** REALLIST

**Description**

The binomial distribution is typically used to predict the number of failures (or defective items or some type of event) in a total of \( n \) independent tests or trials, where each trial has the probability \( p \) of failing or being defective. Each particular test can be considered as a Bernoulli trial.

**num_trials**

- **Keywords Area**
- **variables**
- **binomial_uncertain**
- **num_trials**

A distribution parameter

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Description**

The binomial distribution is typically used to predict the number of failures (or defective items or some type of event) in a total of \( n \) independent tests or trials, where each trial has the probability \( p \) of failing or being defective. Each particular test can be considered as a Bernoulli trial.

**initial_point**

- **Keywords Area**
- **variables**
- **binomial_uncertain**
- **initial_point**

Initial values
6.4. VARIABLES

Specification

Alias: none
Argument(s): INTEGERLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- binomial_uncertain
- descriptors

Labels for the variables

Specification

Alias: none
Argument(s): STRINGLIST
Default: biuv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.22 negative_binomial_uncertain

- Keywords Area
- variables
- negative_binomial_uncertain

Aleatory uncertain discrete variable - negative binomial

Topics

This keyword is related to the topics:

- discrete_variables
- aleatory_uncertain_variables
**Specification**

**Alias:** none  
**Argument(s):** INTEGER  
**Default:** no negative binomial uncertain variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>probability_per_trial</td>
</tr>
<tr>
<td>Required</td>
<td>num_trials</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td>labels for the variables</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The density function for the negative binomial distribution is given by:

\[
f(x) = \binom{n + x - 1}{x} p^n (1 - p)^x
\]

where

- \(p\) is the probability of success per trial
- \(n\) is the number of successful trials
- \(X\) is the number of failures

**Theory**

The negative binomial distribution is typically used to predict the number of failures observed when repeating a test until a total of \(n\) successes have occurred, where each test has a probability \(p\) of success.

**probability_per_trial**

- Keywords Area
- variables
- negative_binomial_uncertain
- probability_per_trial

A negative binomial distribution parameter
6.4. VARIABLES

Specification

Alias: prob_per_trial
Argument(s): REALLIST

Description

The negative binomial distribution is typically used to predict the number of failures observed when repeating a test until a total of \( n \) successes have occurred, where each test has a probability \( p \) of success.

The density function for the negative binomial distribution is given by:

\[
f(x) = \binom{n + x - 1}{x} p^n (1 - p)^x
\]

where

- \( p \) is the probability of success per trial
- \( n \) is the number of successful trials
- \( X \) is the number of failures

num_trials

- Keywords Area
- variables
- negative_binomial_uncertain
- num_trials

A negative binomial distribution parameter

Specification

Alias: none
Argument(s): INTEGERLIST

Description

The negative binomial distribution is typically used to predict the number of failures observed when repeating a test until a total of \( n \) successes have occurred, where each test has a probability \( p \) of success.

The density function for the negative binomial distribution is given by:

\[
f(x) = \binom{n + x - 1}{x} p^n (1 - p)^x
\]

where

- \( p \) is the probability of success per trial
- \( n \) is the number of successful trials
- \( X \) is the number of failures
initial_point

- Keywords Area
- variables
- negative_binomial_uncertain
- initial_point

Initial values

**Specification**

**Alias:** none  
**Argument(s):** INTEGERLIST

**Description**

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- negative_binomial_uncertain
- descriptors

Labels for the variables

**Specification**

**Alias:** none  
**Argument(s):** STRINGLIST  
**Default:** nbuv_{\{i\}}

**Description**

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.23 geometric_uncertain

- Keywords Area
- variables
- geometric_uncertain

Aleatory uncertain discrete variable - geometric
Topics

This keyword is related to the topics:

- discrete_variables
- aleatory_uncertain_variables

Specification

Alias: none

Argument(s): INTEGER

Default: no geometric uncertain variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>probability_per_trial</td>
<td>Geometric distribution parameter</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>initial_point</td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>descriptors</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

The geometric distribution represents the number of successful trials that might occur before a failure is observed.

The density function for the geometric distribution is given by:

\[ f(x) = p(1 - p)^x \]

where \( p \) is the probability of failure per trial.

probability_per_trial

- Keywords Area
- variables
- geometric_uncertain
- probability_per_trial

Geometric distribution parameter

Specification

Alias: prob_per_trial

Argument(s): REALLIST
CHAPTER 6. KEYWORDS AREA

Description
The geometric distribution represents the number of successful trials that occur before a failure is observed. The density function for the geometric distribution is given by:

\[ f(x) = p(1 - p)^x \]

where \( p \) is the probability of failure per trial and \( x \) is the number of successful trials.

**initial_point**
- Keywords Area
- variables
- geometric_uncertain
- initial_point

Initial values

Specification
Alias: none

**Argument(s):** INTEGERLIST

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors
- Keywords Area
- variables
- geometric_uncertain
- descriptors

Labels for the variables

Specification
Alias: none

**Argument(s):** STRINGLIST

**Default:** geuv_{i}

Description
The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.
6.4. VARIABLES

6.4.24 hypergeometric_uncertain

- Keywords Area
- variables
- hypergeometric_uncertain

Aleatory uncertain discrete variable - hypergeometric

Topics

This keyword is related to the topics:

- discrete_variables
- aleatory_uncertain_variables

Specification

Alias: none

Argument(s): INTEGER

Default: no hypergeometric uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>total_population</td>
<td></td>
<td>Parameter for the hypergeometric probability distribution</td>
</tr>
<tr>
<td>Required</td>
<td>selected_population</td>
<td></td>
<td>Distribution parameter for the hypergeometric distribution</td>
</tr>
<tr>
<td>Required</td>
<td>num_drawn</td>
<td></td>
<td>Distribution parameter for the hypergeometric distribution</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

The hypergeometric probability density is used when sampling without replacement from a total population of elements where:

- The resulting element of each sample can be separated into one of two non-overlapping sets
- The probability of success changes with each sample.
The density function for the hypergeometric distribution is given by:

\[ f(x) = \frac{\binom{m}{x} \binom{N - m}{n - x}}{\binom{N}{n}} \]

where:
- \( N \) is the total population
- \( m \) is the number of items in the selected population (e.g. the number of white balls in the full urn of \( N \) items)
- \( n \) is the size of the sample drawn (e.g. number of balls drawn)
- \( x \) is the number of success (e.g. drawing a white ball)
- \( \binom{a}{b} \) is a binomial coefficient

**Theory**

The hypergeometric is often described using an urn model. For example, say we have a total population containing \( N \) balls, and we know that \( m \) of the balls are white and the remaining balls are green. If we draw \( n \) balls from the urn without replacement, the hypergeometric distribution describes the probability of drawing \( x \) white balls.

**total_population**

- **Keywords Area**
- variables
- hypergeometric_uncertain
- total_population

Parameter for the hypergeometric probability distribution

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Description**

The density function for the hypergeometric distribution is given by:

\[ f(x) = \frac{\binom{m}{x} \binom{N - m}{n - x}}{\binom{N}{n}} \]

where
6.4. VARIABLES

- $N$ is the total population (e.g. the total number of balls in the urn)
- $m$ is the number of items in the selected population (e.g. the number of white balls in the full urn of $N$ items)
- $n$ is the size of the sample (e.g. number of balls drawn)
- $x$ is the number of success (e.g. drawing a white ball)
- $\binom{a}{b}$ is a binomial coefficient

**selected_population**

- **Keywords** Area
- **variables**
- **hypergeometric_uncertain**
- **selected_population**

Distribution parameter for the hypergeometric distribution

### Specification

**Alias:** none

**Argument(s):** INTEGERLIST

### Description

The density function for the hypergeometric distribution is given by:

$$f(x) = \binom{m}{x} \binom{N - m}{n - x} \binom{N}{n}$$

where

- $N$ is the total population (e.g. the total number of balls in the urn)
- $m$ is the number of items in the selected population (e.g. the number of white balls in the full urn of $N$ items)
- $n$ is the size of the sample (e.g. number of balls drawn)
- $x$ is the number of success (e.g. drawing a white ball)
- $\binom{a}{b}$ is a binomial coefficient
num_drawn

- Keywords Area
- variables
- hypergeometric_uncertain
- num_drawn

Distribution parameter for the hypergeometric distribution

**Specification**

Alias: none

**Argument(s):** INTEGERLIST

**Description**

The density function for the hypergeometric distribution is given by:

\[
f(x) = \binom{m}{x} \binom{N-m}{n-x} \binom{N}{n}
\]

where

- N is the total population (e.g. the total number of balls in the urn)
- m is the number of items in the selected population (e.g. the number of white balls in the full urn of N items)
- n is the size of the sample (e.g. number of balls drawn)
- x is the number of success (e.g. drawing a white ball)
- \( \binom{a}{b} \) is a binomial coefficient

**initial_point**

- Keywords Area
- variables
- hypergeometric_uncertain
- initial_point

Initial values

**Specification**

Alias: none

**Argument(s):** INTEGERLIST
6.4. VARIABLES

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors
- Keywords Area
- variables
- hypergeometric_uncertain
- descriptors

Labels for the variables

Specification
Alias: none
Argument(s): STRINGLIST
Default: hguv_{i}

Description
The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.25 histogram_point_uncertain
- Keywords Area
- variables
- histogram_point_uncertain

Aleatory uncertain variable - discrete histogram

Topics
This keyword is related to the topics:
- discrete_variables
- aleatory_uncertain_variables

Specification
Alias: none
Argument(s): none
Default: no histogram point uncertain variables
### Description

Histogram uncertain variables are typically used to model a set of empirical data. When the variables take on only discrete values or categories, a discrete, or point histogram is used to describe their probability mass function (one could think of this as a histogram_bin_uncertain variable with "bins" of zero width). Dakota supports integer-, string-, and real-valued point histograms.

Point histograms are similar to discrete_design_set and discrete_state_set, but as they are uncertain variables, include the relative probabilities of observing the different values within the set.

The histogram_point_uncertain keyword is followed by one or more of integer, string, or real, each of which specify the number of variables to be characterized as discrete histograms of that sub-type.

Each discrete histogram variable is specified by one or more abscissa/count pairs. The abscissas, are the possible values the variable can take on ("x" coordinates of type integer, string, or real), and must be specified in increasing order. These are paired with counts \( c \) which provide the frequency of the given value or string, relative to other possible values/strings.

Thus, to fully specify a point-based histogram with \( n \) points, \( n \) \((x,c)\) pairs must be specified with the following features:

- \( x \) is the point value (integer, string, or real) and \( c \) is the corresponding count for that value.
- the \( x \) values must be strictly increasing (lexicographically for strings).
- all \( c \) values must be positive.
- a minimum of one pair must be specified for each point-based histogram.

### Examples

The pairs_per_variable specification provides for the proper association of multiple sets of \((x,c)\) or \((x,y)\) pairs with individual histogram variables. For example, in the following specification,

```
histogram_point_uncertain
  integer = 2
  pairs_per_variable = 2 3
  abscissas = 3 4 100 200 300
  counts = 1 1 1 2 1
```

pairs_per_variable associates the \((x,c)\) pairs \{(3,1),(4,1)\} with one point-based histogram variable (where the values 3 and 4 are equally probable) and associates the \((x,c)\) pairs \{(100,1),(200,2),(300,1)\} with a second point-based histogram variable (where the value 200 is twice as probable as either 100 or 300).
See Also

These keywords may also be of interest:

- `histogram_bin_uncertain`

FAQ

**Difference between bin and point histograms:** A (continuous) bin histogram specifies bins of non-zero width, whereas a (discrete) point histogram specifies individual point values, which can be thought of as bins with zero width. In the terminology of LHS[89], the bin pairs specification defines a "continuous linear" distribution and the point pairs specification defines a "discrete histogram" distribution (although the points are real-valued, the number of possible values is finite).

**integer**

- `Keywords Area`
- `variables`
- `histogram_point_uncertain`
- `integer`

Integer valued point histogram variable

Specification

**Alias:** none

**Argument(s):** INTEGER

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><code>pairs_per_variable</code></td>
<td>Number of pairs defining each histogram point integer variable</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td><code>abscissas</code></td>
<td>Integer abscissas for a point histogram</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td><code>counts</code></td>
<td>Counts for integer-valued point histogram</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>initial_point</th>
<th>Initial values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

This probability mass function is integer-valued; the abscissa values must all be integers. The $n$ abscissa values are paired with $n$ counts which indicate the relative frequency (mass) of each integer relative to the other specified integers.

**Examples**

```plaintext
histogram_point_uncertain
integer = 2
pairs_per_variable = 2 3
abscissas = 3 4 100 200 300
counts = 1 1 1 2 1
```

There are two variables, the first one has two possible integer values which are equally probable. The second one has three options, and 200 is twice as probable as either 100 or 300.

**pairs_per_variable**

- Keywords Area
- variables
- histogram_point_uncertain
- integer
- pairs_per_variable

Number of pairs defining each histogram point integer variable

**Specification**

Alias: num_pairs

Argument(s): INTEGERLIST

Default: equal distribution

**Description**

By default, the list of abscissas and counts will be evenly divided among the histogram point integer variables. The number of pairs_per_variable specifies the apportionment of abscissa/count pairs among the histogram point integer variables. It must specify one integer $\geq 1$ per variable that indicates how many of the (abscissa, count) = (x,c) pairs to associate with that variable.
6.4. VARIABLES

abscissas

- Keywords Area
- variables
- histogram_point_uncertain
- integer
- abscissas

Integer abscissas for a point histogram

Specification

Alias: none

Description

A list of integer abscissa ("x" coordinate) values characterizing the probability density function for each of the integer histogram_point_uncertain variables. These must be listed in increasing order for each variable, and are paired with counts. See histogram_point_uncertain for details and examples.

counts

- Keywords Area
- variables
- histogram_point_uncertain
- integer
- counts

Counts for integer-valued point histogram

Specification

Alias: none

Description

Count or frequency for each of abscissas. See histogram_point_uncertain for details and examples.
initial_point

- Keywords Area
- variables
- histogram_point_uncertain
- integer
- initial_point

Initial values

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Description**

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- histogram_point_uncertain
- integer
- descriptors

Labels for the variables

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** hpiv_{i}

**Description**

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.
### 6.4. VARIABLES

**string**

- **Keywords Area**
- **variables**
- **histogram_point_uncertain**
- **string**

String (categorical) valued point histogram variable

### Specification

**Alias:** none

**Argument(s):** INTEGER

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>pairs_per_variable</td>
<td>Number of pairs defining each histogram point string variable</td>
</tr>
<tr>
<td>Required</td>
<td>abscissas</td>
<td></td>
<td>String abscissas for a point histogram</td>
</tr>
<tr>
<td>Required</td>
<td>counts</td>
<td></td>
<td>Counts for string-valued point histogram</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

### Description

This probability mass function is string-valued; the abscissa values must all be strings. The \( n \) abscissa values are paired with \( n \) counts which indicate the relative frequency (mass) of each string relative to the other specified strings.

### Examples

```plaintext
histogram_point_uncertain
string = 2
pairs_per_variable = 2 3
abscissas = 'no' 'yes' 'function1' 'function2' 'function3'
counts = 1 1 1 2 1
descriptors = 'vote' 'which_function'
```

Here there are two variables, the first one ('vote') has two possible string values 'yes' and 'no' which are equally probable. The second one has three options for 'which_function', and 'function2' is twice as probable as 'function1' or 'function3'.
pairs_per_variable

- Keywords Area
- variables
- histogram_point_uncertain
- string
- pairs_per_variable

Number of pairs defining each histogram point string variable

**Specification**

Alias: num_pairs

- **Argument(s):** INTEGERLIST
- **Default:** equal distribution

**Description**

By default, the list of abscissas and counts will be evenly divided among the histogram point string variables. The number of pairs per variable specifies the apportionment of abscissa/count pairs among the histogram point string variables. It must specify one integer $\geq 1$ per variable that indicates how many of the (abscissa, count) = (x,c) pairs to associate with that variable.

abscissas

- Keywords Area
- variables
- histogram_point_uncertain
- string
- abscissas

String abscissas for a point histogram

**Specification**

Alias: none

- **Argument(s):** STRINGLIST

**Description**

A list of string abscissa ("x" coordinate) values characterizing the probability density function for each of the string histogram_point_uncertain variables. These must be listed in (lexicographically) increasing order for each variable, and are paired with counts. See histogram_point_uncertain for details and examples.
counts

- Keywords Area
- variables
- histogram_point_uncertain
- string
- counts

Counts for string-valued point histogram

**Specification**

**Alias:** none  
**Argument(s):** REALLIST

**Description**

Count or frequency for each of abscissas. See histogram_point_uncertain for details and examples.

**initial_point**

- Keywords Area
- variables
- histogram_point_uncertain
- string
- initial_point

Initial values

**Specification**

**Alias:** none  
**Argument(s):** STRINGLIST

**Description**

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).
CHAPTER 6. KEYWORDS AREA

descriptors

- Keywords Area
- variables
- histogram_point_uncertain
- string
- descriptors

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: hpsv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

real

- Keywords Area
- variables
- histogram_point_uncertain
- real

Real valued point histogram variable

Specification

Alias: none

Argument(s): INTEGER

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional   | Group          | Description    | Description   |
| Optional   |               | of Group       | Description   |
|            |                | pairs_per_variable | Number of pairs defining each histogram point |
|            |                |                | real variable |
6.4. VARIABLES

<table>
<thead>
<tr>
<th>Required</th>
<th>abscissas</th>
<th>Real abscissas for a point histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>counts</td>
<td>Counts for real-valued point histogram</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

**Description**

This probability mass function is real-valued; the abscissa values must all be integers. The \( n \) abscissa values are paired with \( n \) counts which indicate the relative frequency (mass) of each real relative to the other specified reals.

**Examples**

```
histogram_point_uncertain
  real = 2
  pairs_per_variable = 2  3
  abscissas = 3.1415 4.5389 100 200.112345 300
  counts = 1 1 1 2 1
```

There are two variables, the first one has two possible real values which are equally probable. The second one has three possible real value options, and 200.112345 is twice as probable as either 100 or 300.

**pairs_per_variable**

- **Keywords** Area
- **variables**
- **histogram_point_uncertain**
- **real**
- **pairs_per_variable**

Number of pairs defining each histogram point real variable

**Specification**

**Alias:** num_pairs

**Argument(s):** INTEGERLIST

**Default:** equal distribution
**Description**

By default, the list of **abscissas** and **counts** will be evenly divided among the histogram point real variables. The number of **pairs_per_variable** specifies the apportionment of abscissa/count pairs among the histogram point real variables. It must specify one integer \( \geq 1 \) per variable that indicates how many of the (abscissa, count) \( = (x, c) \) pairs to associate with that variable.

**abscissas**
- Keywords Area
- variables
- histogram_point_uncertain
- real
- abscissas

Real abscissas for a point histogram

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Description**

A list of real abscissa ("x" coordinate) values characterizing the probability density function for each of the real **histogram_point_uncertain** variables. These must be listed in increasing order for each variable, and are paired with **counts**. See **histogram_point_uncertain** for details and examples.

**counts**
- Keywords Area
- variables
- histogram_point_uncertain
- real
- counts

Counts for real-valued point histogram

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Description**

Count or frequency for each of **abscissas**. See **histogram_point_uncertain** for details and examples.
6.4. VARIABLES

**initial_point**

- Keywords Area
- variables
- histogram_point_uncertain
- real
- initial_point

Initial values

**Specification**

**Alias:** none

**Argument(s):** REALLIST

**Description**

The `initial_point` specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

**descriptors**

- Keywords Area
- variables
- histogram_point_uncertain
- real
- descriptors

Labels for the variables

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** hpruv.{i}

**Description**

The optional variables labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.
6.4.26 uncertain_correlation_matrix

- Keywords Area
- variables
  - uncertain_correlation_matrix

Correlation among aleatory uncertain variables

Specification

Alias: none
- Argument(s): REALLIST
- Default: identity matrix (uncorrelated)

Description

Aleatory uncertain variables may have correlations specified through use of an uncertain_correlation_matrix specification. This specification is generalized in the sense that its specific meaning depends on the nondeterministic method in use.

When the method is a nondeterministic sampling method (i.e., sampling), then the correlation matrix specifies rank correlations [52].

When the method is a reliability (i.e., local_reliability or global_reliability) or stochastic expansion (i.e., polynomial_choas or stoch_collocation) method, then the correlation matrix specifies correlation coefficients (normalized covariance) [42].

In either of these cases, specifying the identity matrix results in uncorrelated uncertain variables (the default). The matrix input should be symmetric and have all $n^2$ entries where $n$ is the total number of aleatory uncertain variables.

Ordering of the aleatory uncertain variables is:

1. normal
2. lognormal
3. uniform
4. loguniform
5. triangular
6. exponential
7. beta
8. gamma
9. gumbel
10. frechet
11. weibull
12. histogram bin
13. poisson
14. binomial
15. negative binomial
16. geometric
17. hypergeometric
18. histogram point

When additional variable types are activated, they assume uniform distributions, and the ordering is as listed on variables.

Examples

Consider the following random variables, distributions and correlations:

- $X_1$, normal, uncorrelated with others
- $X_2$, normal, correlated with $X_3$, $X_4$ and $X_5$
- $X_3$, weibull, correlated with $X_5$
- $X_4$, exponential, correlated with $X_3$, $X_4$ and $X_5$
- $X_5$, normal, correlated with $X_5$

These correlations are captured by the following commands (order of the variables is respected).

```plaintext
uncertain_correlation_matrix
# ordering normal, exponential, weibull
# $X_1\ $ $X_2\ $ $X_3\ $ $X_4\ $ $X_5$
1.00 0.00 0.00 0.00 0.00
0.00 1.00 0.50 0.24 0.78
0.00 0.50 1.00 0.00 0.20
0.00 0.24 0.00 1.00 0.49
0.00 0.78 0.20 0.49 1.0
```

6.4.27 continuous_interval_uncertain

- **Keywords** Area
- **variables**
- **continuous_interval_uncertain**

Epistemic uncertain variable - values from one or more continuous intervals

Topics

This keyword is related to the topics:

- **continuous_variables**
- **epistemic_uncertain_variables**

Specification

**Alias:** interval_uncertain
**Argument(s):** INTEGER
**Default:** no continuous interval uncertain variables
### Description

Continuous interval uncertain variables are epistemic types. They can specify a single interval per variable which may be used in interval analysis, where the goal is to determine the interval bounds on the output corresponding to the interval bounds on the input. All values between the bounds are permissible. More detailed continuous interval representations can specify a set of belief structures based on intervals that may be contiguous, overlapping, or disjoint. This is used in specifying the inputs necessary for an epistemic uncertainty analysis using Dempster-Shafer theory of evidence.

Other epistemic types include:

- `discrete_interval_uncertain`
- `discrete_uncertain_set integer`
- `discrete_uncertain_set string`
- `discrete_uncertain_set real`

### Examples

The following specification is for an interval analysis:

```plaintext
continuous_interval_uncertain = 2
lower_bounds = 2.0 4.0
upper_bounds = 2.5 5.0
```

The following specification is for a Dempster-Shafer analysis:

```plaintext
continuous_interval_uncertain = 2
num_intervals = 3, 2
interval_probs = 0.25 0.5 0.25 0.4 0.6
lower_bounds = 2.0 4.0 4.5 1.0 3.0
upper_bounds = 2.5 5.0 6.0 5.0 5.0
```

Here there are 2 interval uncertain variables. The first one is defined by three intervals, and the second by two intervals. The three intervals for the first variable have basic probability assignments of 0.2, 0.5, and 0.3, respectively, while the basic probability assignments for the two intervals for the second variable are 0.4 and 0.6.
The basic probability assignments for each interval variable must sum to one. The interval bounds for the first variable are [2, 2.5], [4, 5], and [4.5, 6], and the interval bounds for the second variable are [1.0, 5.0] and [3.0, 5.0]. Note that the intervals can be overlapping or disjoint. The BPA for the first variable indicates that it is twice as likely that the value occurs on the interval [4,5] than either [2,2.5] or [4.5,6].

Theory

The continuous interval uncertain variable is NOT a probability distribution. Although it may seem similar to a histogram, the interpretation of this uncertain variable is different. It is used in epistemic uncertainty analysis, where one is trying to model uncertainty due to lack of knowledge. The continuous interval uncertain variable is used in both interval analysis and in Dempster-Shafer theory of evidence.

- **interval analysis** -only one interval is allowed for each *continuous_interval_uncertain* variable -the interval is defined by lower and upper bounds -the value of the random variable lies somewhere in this interval -output is the minimum and maximum function value conditional on the specified interval

- **Dempster-Shafer theory of evidence** -multiple intervals can be assigned to each *continuous_interval_uncertain* variable -a Basic Probability Assignment (BPA) is associated with each interval. The BPA represents a probability that the value of the uncertain variable is located within that interval. -each interval is defined by lower and upper bounds -outputs are called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the intervals together with their BPA.

**num_intervals**

- **Keywords Area**
- **variables**
- **continuous_interval_uncertain**
- **num_intervals**

Specify the number of intervals for each variable

**Specification**

**Alias:** iuv_num_intervals

**Argument(s):** INTEGERLIST

**Default:** Equal apportionment of intervals among variables

**Description**

In Dakota, epistemic uncertainty analysis is performed using either interval estimation or Dempster-Shafer theory of evidence. In these approaches, one does not assign a probability distribution to each uncertain input variable. Rather, one divides each uncertain input variable into one or more intervals. The input parameters are only known to occur within intervals; nothing more is assumed. *num_intervals* specifies the number of such intervals associated with each interval uncertain parameter.
interval_probabilities

- Keywords Area
- variables
- continuous_interval_uncertain
- interval_probabilities

Assign probability mass to each interval

**Specification**

**Alias:** interval_probs iuv_interval_probs  
**Argument(s):** REALLIST  
**Default:** Equal probability assignments for each interval \(1/\text{num_intervals}[i]\)

**Description**

The basic probability assignments for each interval variable must sum to one. For example, if an interval variable is defined with three intervals, the probabilities for these intervals could be 0.2, 0.5, and 0.3 which sum to one, but could not be 0.5,0.5, and 0.5 which do not sum to one.

lower_bounds

- Keywords Area
- variables
- continuous_interval_uncertain
- lower_bounds

Specify minimum values

**Specification**

**Alias:** none  
**Argument(s):** REALLIST

**Description**

Specify minimum values

upper_bounds

- Keywords Area
- variables
- continuous_interval_uncertain
- upper_bounds

Specify maximum values
6.4. VARIABLES

**Specification**

*Alias:* none

*Argument(s):* REALLIST

**Description**

Specify maximum values

**initial_point**

- Keywords Area
- variables
- `continuous_interval_uncertain`
- `initial_point`

Initial values

**Specification**

*Alias:* none

*Argument(s):* REALLIST

**Description**

The `initial_point` specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

**descriptors**

- Keywords Area
- variables
- `continuous_interval_uncertain`
- descriptors

Labels for the variables

**Specification**

*Alias:* `iuv_descriptors`

*Argument(s):* STRINGLIST

*Default:* `ciuv_{i}`
CHAPTER 6. KEYWORDS AREA

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output. The default descriptor strings use a root string plus a numeric identifier.

6.4.28 discrete_interval_uncertain

• Keywords Area
• variables
• discrete_interval_uncertain

Epistemic uncertain variable - values from one or more discrete intervals

Topics

This keyword is related to the topics:

• discrete_variables
• epistemic_uncertain_variables

Specification

Alias: discrete_uncertain_range

Argument(s): INTEGER

Default: No discrete interval uncertain variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>num_intervals</td>
<td>Specify the number of intervals for each variable</td>
</tr>
<tr>
<td>Optional</td>
<td>interval_probabilities</td>
<td></td>
<td>Assign probability mass to each interval</td>
</tr>
<tr>
<td>Required</td>
<td>lower_bounds</td>
<td></td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Required</td>
<td>upper_bounds</td>
<td></td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td></td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

Discrete interval uncertain variables are epistemic types. They can specify a single interval per variable which may be used in interval analysis, where the goal is to determine the interval bounds on the output corresponding to the interval bounds on the input. Permissible values are any integer within the bound. More detailed continuous...
interval representations can specify a set of belief structures based on intervals that may be contiguous, overlapping, or disjoint. This is used in specifying the inputs necessary for an epistemic uncertainty analysis using Dempster-Shafer theory of evidence.

Other epistemic types include:

- `continuous_interval_uncertain`
- `discrete_uncertain_set integer`
- `discrete_uncertain_set string`
- `discrete_uncertain_set real`

**Examples**

Let \( d_1 \) be 2, 3 or 4 with probability 0.2, 4 or 5 with probability 0.5 and 6 with probability 0.3. Let \( d_2 \) be 4, 5 or 6 with probability 0.4 and 6, 7 or 8 with probability 0.6. The following specification is for a Dempster-Shafer analysis:

```plaintext
discrete_interval_uncertain = 2
num_intervals = 3 2
interval_probs = 0.2 0.5 0.3 0.4 0.6
lower_bounds = 2 4 6 4 6
upper_bounds = 4 5 6 6 8
```

**Theory**

- Dempster-Shafer theory of evidence -multiple intervals can be assigned to each `discrete_interval_uncertain` variable -a Basic Probability Assignment (BPA) is associated with each interval. The BPA represents a probability that the value of the uncertain variable is located within that interval. -each interval is defined by lower and upper bounds -outputs are called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the intervals together with their BPA.

**num_intervals**

- **Keywords**: Area
- **variables**
- `discrete_interval_uncertain`
- **num_intervals**

Specify the number of intervals for each variable

**Specification**

**Alias**: none

**Argument(s)**: INTEGERLIST

**Default**: Equal apportionment of intervals among variables
**Description**

In Dakota, epistemic uncertainty analysis is performed using either interval estimation or Dempster-Shafer theory of evidence. In these approaches, one does not assign a probability distribution to each uncertain input variable. Rather, one divides each uncertain input variable into one or more intervals. The input parameters are only known to occur within intervals; nothing more is assumed. `num_intervals` specifies the number of such intervals associated with each interval uncertain parameter.

**interval_probabilities**

- Keywords Area
- variables
- `discrete_interval_uncertain`
- `interval_probabilities`

Assign probability mass to each interval

**Specification**

**Alias:** interval_probs range_probabilities range_probs

**Argument(s):** REALLIST

**Default:** Equal probability assignments for each interval \((1/\text{num\_intervals}[i])\)

**Description**

The basic probability assignments for each interval variable must sum to one. For example, if an interval variable is defined with three intervals, the probabilities for these intervals could be 0.2, 0.5, and 0.3 which sum to one, but could not be 0.5, 0.5, and 0.5 which do not sum to one.

**lower_bounds**

- Keywords Area
- variables
- `discrete_interval_uncertain`
- `lower_bounds`

Specify minimum values

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Description**

Specify minimum values
6.4. VARIABLES

**upper_bounds**
- **Keywords Area**
- **variables**
- **discrete_interval_uncertain**
- **upper_bounds**

Specify maximum values

**Specification**
Alias: none
Argument(s): INTEGRALIST

**Description**
Specify maximum values

**initial_point**
- **Keywords Area**
- **variables**
- **discrete_interval_uncertain**
- **initial_point**

Initial values

**Specification**
Alias: none
Argument(s): INTEGRALIST

**Description**
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

**descriptors**
- **Keywords Area**
- **variables**
- **discrete_interval_uncertain**
- **descriptors**

Labels for the variables
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRINGLIST

Default: diuv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.29 discrete_uncertain_set

- Keywords Area
- variables
- discrete_uncertain_set

Set-valued discrete uncertain variables

Topics

This keyword is related to the topics:

- discrete_variables
- epistemic_uncertain_variables

Specification

Alias: none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>integer</td>
<td>Discrete, epistemic uncertain variable - integers within a set</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>string</td>
<td>Discrete, epistemic uncertain variable - strings within a set</td>
</tr>
</tbody>
</table>
6.4. VARIABLES

### Description

Discrete uncertain variables whose values come from a set of admissible elements. Each variable specified must be of type integer, string, or real.

**integer**

- **Keywords** Area
- variables
- discrete_uncertain_set
- integer

Discrete, epistemic uncertain variable - integers within a set

### Topics

This keyword is related to the topics:

- discrete_variables
- epistemic_uncertain_variables

### Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** no discrete uncertain set integer variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>real</td>
</tr>
<tr>
<td></td>
<td>Discrete, epistemic uncertain variable - real numbers within a set</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elements_per_variable</td>
<td></td>
</tr>
<tr>
<td>elements</td>
<td></td>
</tr>
</tbody>
</table>

- The permissible values for each discrete variable

- Number of admissible elements for each set variable
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>set_probabilities</th>
<th>This keyword defines the probabilities for the various elements of discrete sets. Whether the set-valued variables are categorical or relaxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>categorical</td>
<td>Initial values</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_point</td>
<td>Labels for the variables</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Discrete set variables may be used to specify categorical choices which are epistemic. For example, if we have three possible forms for a physics model (model 1, 2, or 3) and there is epistemic uncertainty about which one is correct, a discrete uncertain set may be used to represent this type of uncertainty.

This variable is defined by a set of integers, in which the discrete value may take any value within the integer set (for example, the set may be defined as 1, 2, and 4).

Other epistemic types include:

- `continuous_interval_uncertain`
- `discrete_interval_uncertain`
- `discrete_uncertain_set string`
- `discrete_uncertain_set real`

**Examples**

Let d1 be 2 or 13 and d2 be 4, 5 or 26. The following specification is for an interval analysis:

```plaintext
discrete_uncertain_set
integer
num_set_values 2 3
set_values 2 13 4 5 26
descriptors 'di1' 'di2'
```

**Theory**

The `discrete_uncertain_set_integer` variable is NOT a discrete random variable. It can be contrasted to the histogram-defined random variables: `histogram_bin_uncertain` and `histogram_point_uncertain`. It is used in epistemic uncertainty analysis, where one is trying to model uncertainty due to lack of knowledge.

The discrete uncertain set integer variable is used in both interval analysis and in Dempster-Shafer theory of evidence.
• interval analysis - the values are integers, equally weighted - the true value of the random variable is one of the integers in this set - output is the minimum and maximum function value conditional on the specified inputs

• Dempster-Shafer theory of evidence - the values are integers, but they can be assigned different weights - outputs are called “belief” and “plausibility.” Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the values together with their weights.

**elements_per_variable**
- Keywords Area
- variables
- discrete_uncertain_set
- integer
- elements_per_variable

Number of admissible elements for each set variable

**Specification**
*Alias:* num_set_values

**Argument(s):** INTEGERLIST

**Default:** equal distribution

**Description**
Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

**elements**
- Keywords Area
- variables
- discrete_uncertain_set
- integer
- elements

The permissible values for each discrete variable

**Specification**
*Alias:* set_values

**Argument(s):** INTEGERLIST
CHAPTER 6. KEYWORDS AREA

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete_variables page.

set_probabilities

- Keywords Area
- variables
- discrete_uncertain_set
- integer
- set_probabilities

This keyword defines the probabilities for the various elements of discrete sets.

Specification

Alias: set_probs

Argument(s): REALLIST

Default: Equal probability assignments for each set member (1/num_set_values[i])

Description

There are three types of discrete_uncertain_set variables: integer, string, or real sets. With each of these types, one defines the number of elements of the set per that variable, the values of those elements, and the associated probabilities. For example, if one has an integer discrete uncertain set variable with 3 elements {3,4,8}, then one could define the probabilities associated with those set elements as (for example) 0.2, 0.5, and 0.3. The set_probabilities for a particular variable should sum to one over all the elements in that set.

categorical

- Keywords Area
- variables
- discrete_uncertain_set
- integer
- categorical

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none

Argument(s): STRINGLIST
6.4. VARIABLES

Description
A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include ’yes’, ’no’, ’true’, and ’false’, or any abbreviation in [yYnNtTfF].

Examples
Discrete design set variable, ’rotor blades’, can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and ’rotor blades’ can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```
 discrete_design_set
   integer 1
   elements 2 4 7
   descriptor ’rotor_blades’
   categorical ’no’
```

initial_point

- Keywords Area
- variables
- discrete_uncertain_set
- integer
- initial_point

Initial values

Specification

Alias: none

Argument(s): INTEGERLIST

Description
The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- discrete_uncertain_set
- integer
- descriptors

Labels for the variables
CHAPTER 6. KEYWORDS AREA

**Specification**

Alias: none

**Argument(s):** STRINGLIST

**Default:** dusiv_{i}

**Description**

The optional variables labels specification *descriptors* is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

**string**

- **Keywords Area**
- **variables**
- **discrete_uncertain_set**
- **string**

Discrete, epistemic uncertain variable - strings within a set

**Topics**

This keyword is related to the topics:

- **discrete_variables**
- **epistemic_uncertain_variables**

**Specification**

Alias: none

**Argument(s):** INTEGER

**Default:** no discrete uncertain set string variables

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>elements_per-variable</td>
<td>Number of admissible elements for each set variable</td>
</tr>
</tbody>
</table>
### Description

Discrete set variables may be used to specify categorical choices which are epistemic. For example, if we have three possible forms for a physics model (model 1, 2, or 3) and there is epistemic uncertainty about which one is correct, a discrete uncertain set may be used to represent this type of uncertainty.

This variable is defined by a set of strings, in which the discrete value may take any value within the string set (for example, the set may be defined as ‘coarse’, ‘medium’, and ‘fine’)

Other epistemic types include:

- `continuous_interval_uncertain`
- `discrete_interval_uncertain`
- `discrete_uncertain_set integer`
- `discrete_uncertain_set real`

### Examples

```plaintext
 discrete_uncertain_set
 string
 num_set_values 2 3
 set_values ‘red’ ‘blue’ ‘coarse’ ‘medium’ ‘fine’
 descriptors ‘ds1’ ‘ds2’
```

#### elements_per_variable

- Keywords: `Area`
- `variables`
- `discrete_uncertain_set`
- `string`
- `elements_per_variable`

Number of admissible elements for each set variable
CHAPTER 6. KEYWORDS AREA

Specification

Alias: num_set_values
Argument(s): INTEGERLIST
Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

elements

- Keywords Area
- variables
- discrete_uncertain_set
- string
- elements

The permissible values for each discrete variable

Specification

Alias: set_values
Argument(s): STRINGLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete_variables page.

set_probabilities

- Keywords Area
- variables
- discrete_uncertain_set
- string
- set_probabilities

This keyword defines the probabilities for the various elements of discrete sets.
6.4. VARIABLES

Specification

Alias: set_probs

Argument(s): REALLIST

Default: Equal probability assignments for each set member (1/num_set_values[i])

Description

There are three types of discrete uncertain set variables: integer, string, or real sets. With each of these types, one defines the number of elements of the set per that variable, the values of those elements, and the associated probabilities. For example, if one has an integer discrete uncertain set variable with 3 elements \( \{3, 4, 8\} \), then one could define the probabilities associated with those set elements as (for example) 0.2, 0.5, and 0.3. The set probabilities for a particular variable should sum to one over all the elements in that set.

initial_point

- Keywords Area
- variables
- discrete_uncertain_set
- string
- initial_point

Initial values

Specification

Alias: none

Argument(s): STRINGLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).

descriptors

- Keywords Area
- variables
- discrete_uncertain_set
- string
- descriptors

Labels for the variables
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): STRINGLIST
Default: duvsv, \{i\}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output. The default descriptor strings use a root string plus a numeric identifier.

real
• Keywords Area
• variables
• discrete_uncertain_set
• real

Discrete, epistemic uncertain variable - real numbers within a set

Topics

This keyword is related to the topics:
• discrete_variables
• epistemic_uncertain_variables

Specification

Alias: none
Argument(s): INTEGER
Default: no discrete uncertain set real variables

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>elements_per_variable</td>
<td>Number of admissible elements for each set variable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>elements_per_variable</td>
<td>Number of admissible elements for each set variable</td>
</tr>
</tbody>
</table>
Description

Discrete set variables may be used to specify categorical choices which are epistemic. For example, if we have three possible forms for a physics model (model 1, 2, or 3) and there is epistemic uncertainty about which one is correct, a discrete uncertain set may be used to represent this type of uncertainty.

This variable is defined by a set of reals, in which the discrete variable may take any value defined within the real set (for example, a parameter may have two allowable real values, 3.285 or 4.79).

Other epistemic types include:

- continuous_interval_uncertain
- discrete_interval_uncertain
- discrete_uncertain_set integer
- discrete_uncertain_set string

Examples

Let d1 be 2.1 or 1.3 and d2 be 0.4, 5 or 2.6. The following specification is for an interval analysis:

```plaintext
discrete_uncertain_set
integer
num_set_values 2 3
set_values 2.1 1.3 0.4 5 2.6
descriptors 'dr1' 'dr2'
```

Theory

The `discrete_uncertain_set-integer` variable is NOT a discrete random variable. It can be contrasted to the histogram-defined random variables: `histogram_bin_uncertain` and `histogram_point_uncertain`. It is used in epistemic uncertainty analysis, where one is trying to model uncertainty due to lack of knowledge.
The discrete uncertain set integer variable is used in both interval analysis and in Dempster-Shafer theory of evidence.

- **interval analysis**: the values are integers, equally weighted -the true value of the random variable is one of the integers in this set -output is the minimum and maximum function value conditional on the specified inputs

- **Dempster-Shafer theory of evidence**: the values are integers, but they can be assigned different weights -outputs are called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the values together with their weights.

**elements_per_variable**

- **Keywords Area**
- **variables**
- **discrete_uncertain_set**
- **real**
- **elements_per_variable**

Number of admissible elements for each set variable

**Specification**

**Alias**: num_set_values

**Argument(s)**: INTEGERLIST

**Default**: equal distribution

**Description**

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

**elements**

- **Keywords Area**
- **variables**
- **discrete_uncertain_set**
- **real**
- **elements**

The permissible values for each discrete variable
6.4. VARIABLES

Specification

Alias: set_values

Argument(s): REALLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete_variables page.

set_probabilities

- Keywords Area

- variables

- discrete_uncertain_set

- real

- set_probabilities

This keyword defines the probabilities for the various elements of discrete sets.

Specification

Alias: set_probs

Argument(s): REALLIST

Default: Equal probability assignments for each set member (1/num_set_values[i])

Description

There are three types of discrete_uncertain_set variables: integer, string, or real sets. With each of these types, one defines the number of elements of the set per that variable, the values of those elements, and the associated probabilities. For example, if one has an integer discrete uncertain set variable with 3 elements \{3,4,8\}, then one could define the probabilities associated with those set elements as (for example) 0.2, 0.5, and 0.3. The set_probabilities for a particular variable should sum to one over all the elements in that set.

categorical

- Keywords Area

- variables

- discrete_uncertain_set

- real

- categorical

Whether the set-valued variables are categorical or relaxable
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): STRINGLIST

Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

discrete_design_set
  integer 1
  elements 2 4 7
descriptor 'rotor_blades'
categorical 'no'

Initial_point

• Keywords Area

• variables

• discrete_uncertain_set

• real

• initial_point

Initial values

Specification

Alias: none

Argument(s): REALLIST

Description

The initial_point specifications provide the point in design space (variable values) from which an iterator is started. These default to the midpoint of bounds (continuous design variables) or the middle value (discrete design variables).
6.4. VARIABLES

Descriptors

- Keywords Area
- variables
- discrete_uncertain_set
- real
- descriptors

Labels for the variables

Specification

Alias: none
Argument(s): STRINGLIST
Default: dusrv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.30 continuous_state

- Keywords Area
- variables
- continuous_state

Continuous state variables

Topics

This keyword is related to the topics:

- state_variables
- continuous_variables

Specification

Alias: none
Argument(s): INTEGER
Default: No continuous state variables

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Description

Continuous state variables are defined by bounds.
    Default behavior for most methods is that only the initial_state values are used.
    See the state_variables page for details on the behavior of state variables.

initial_state

- Keywords Area
- variables
- continuous_state
- initial_state

Initial values for the state variables

Specification

Alias: csv_initial_state
Argument(s): REALLIST
Default: 0.0

Description

The initial_state specifications provide the initial values for the state variables.
    This is an optional keyword. If it is not specified, the initial state will be inferred from the other keywords that define the state variable.
    Defaults are:

- Continuous state variables - use the midpoint of the bounds
- Set variables - use the value with the index closest to the middle of the set
- Range variables - use the midpoint of the range

<table>
<thead>
<tr>
<th>Optional</th>
<th>initial_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>lower_bounds</td>
</tr>
<tr>
<td>Optional</td>
<td>upper_bounds</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
</tr>
</tbody>
</table>
6.4. VARIABLES

lower_bound
- Keywords Area
- variables
- continuous_state
- lower_bounds

Specify minimum values

Specification
Alias: csv_lower_bounds
Argument(s): REALLIST
Default: -infinity

Description
Specify minimum values

upper_bound
- Keywords Area
- variables
- continuous_state
- upper_bounds

Specify maximum values

Specification
Alias: csv_upper_bounds
Argument(s): REALLIST
Default: infinity

Description
Specify maximum values

descriptors
- Keywords Area
- variables
- continuous_state
- descriptors

Labels for the variables
CHAPTER 6. KEYWORDS AREA

Specification

Alias: csv_descriptors
  Argument(s): STRINGLIST
  Default: csv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.31 discrete_state_range

- Keywords Area
- variables
- discrete_state_range

Discrete state variables; each defined by an integer interval

Topics

This keyword is related to the topics:

- discrete_variables
- state_variables

Specification

Alias: none
  Argument(s): INTEGER
  Default: No discrete state variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>initial_state</td>
<td>Initial values for the state variables</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>lower_bounds</td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>upper_bounds</td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>descriptors</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

Description

Discrete state variables defined by bounds.

The details of how to specify this discrete variable are located on the discrete_variables page.

See the state_variables page for details on the behavior of state variables.
6.4. VARIABLES

initial_state

- Keywords Area
- variables
- discrete_state_range
- initial_state

Initial values for the state variables

Specification

Alias: dsv_initial_state

Argument(s): INTEGERLIST

Default: 0

Description

The initial_state specifications provide the initial values for the state variables.
This is an optional keyword. If it is not specified, the initial state will be inferred from the other keywords that define the state variable.
Defaults are:

- Continuous state variables - use the midpoint of the bounds
- Set variables - use the value with the index closest to the middle of the set
- Range variables - use the midpoint of the range

lower_bounds

- Keywords Area
- variables
- discrete_state_range
- lower_bounds

Specify minimum values

Specification

Alias: dsv_lower_bounds

Argument(s): INTEGERLIST

Default: INT_MIN

Description

Specify minimum values
upper_bounds

- Keywords Area
- variables
- discrete_state_range
- upper_bounds

Specify maximum values

Specification

Alias: dsv_upper_bounds
Argument(s): INTEGERLIST
Default: INT_MAX

Description

Specify maximum values

descriptors

- Keywords Area
- variables
- discrete_state_range
- descriptors

Labels for the variables

Specification

Alias: dsv_descriptors
Argument(s): STRINGLIST
Default: dsvriv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.4.32 discrete_state_set

- Keywords Area
- variables
- discrete_state_set

Set-valued discrete state variables
6.4. VARIABLES

Topics
This keyword is related to the topics:

- discrete_variables
- state_variables

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>Group</td>
<td>integer</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>string</td>
<td>Discrete state variables, each defined by a set of permissible integers</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>real</td>
<td>String-valued discrete state set variables</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td></td>
<td>Discrete state variables, each defined by a set of permissible real numbers</td>
</tr>
</tbody>
</table>

Description
Discrete state variables whose values come from a set of admissible elements. Each variable specified must be of type integer, string, or real.

integer
- Keywords Area
- variables
- discrete_state_set
- integer

Discrete state variables, each defined by a set of permissible integers

Topics
This keyword is related to the topics:

- discrete_variables
- state_variables
### Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** no discrete set integer variables

---

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>elements_per_variable</td>
<td>Number of admissible elements for each set variable</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td>elements</td>
<td>The permissible values for each discrete variable</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>categorical</td>
<td>Whether the set-valued variables are categorical or relaxable</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>initial_state</td>
<td>Initial values for the state variables</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>descriptors</td>
<td>Labels for the variables</td>
</tr>
</tbody>
</table>

### Description

Discrete state variables defined by a set of permissible integers.

The details of how to specify this discrete variable are located on the `discrete_variables` page.

See the `state_variables` page for details on the behavior of state variables.

#### elements_per_variable

- Keywords Area
- variables
- discrete_state_set
- integer
- elements_per_variable

Number of admissible elements for each set variable

### Specification

**Alias:** num_set_values

**Argument(s):** INTEGERLIST

**Default:** equal distribution
6.4. VARIABLES

Description
Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

elements
- Keywords Area
- variables
- discrete_state_set
- integer
- elements

The permissible values for each discrete variable

Specification
Alias: set_values
- Argument(s): INTEGERLIST

Description
Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete_variables page.

categorical
- Keywords Area
- variables
- discrete_state_set
- integer
- categorical

Whether the set-valued variables are categorical or relaxable

Specification
Alias: none
- Argument(s): STRINGLIST

Description
A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*
Examples
Discrete design set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

discrete_design_set
  integer 1
  elements 2 4 7
descriptor 'rotor_blades'
categorical 'no'

initial_state
  • Keywords Area
  • variables
  • discrete_state_set
  • integer
  • initial_state

Initial values for the state variables

Specification
Alias: none
Argument(s): INTEGERLIST
Default: middle set value, or rounded down

Description
The initial_state specifications provide the initial values for the state variables.
This is an optional keyword. If it is not specified, the initial state will be inferred from the other keywords that define the state variable.
Defaults are:
  • Continuous state variables - use the midpoint of the bounds
  • Set variables - use the value with the index closest to the middle of the set
  • Range variables - use the midpoint of the range

descriptors
  • Keywords Area
  • variables
  • discrete_state_set
  • integer
  • descriptors

Labels for the variables
6.4. VARIABLES

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** \(dssiv\{i\}\)

**Description**

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

**string**

- **Keywords Area**
- **variables**
- **discrete_state_set**
- **string**

String-valued discrete state set variables

**Topics**

This keyword is related to the topics:

- **discrete_variables**
- **state_variables**

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** no discrete state set string variables

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>elements_per_variable</td>
<td>Number of admissible elements for each set variable</td>
</tr>
</tbody>
</table>
Required  |  elements  | The permissible values for each discrete variable
Optional  |  initial_state  | Initial values for the state variables
Optional  |  descriptors  | Labels for the variables

**Description**

Discrete state variables whose values come from a specified set of admissible strings. The details of how to specify this discrete variable are located on the `discrete_variables` page. See the `state_variables` page for details on the behavior of state variables. Each string element value must be quoted and may contain alphanumeric, dash, underscore, and colon. White space, quote characters, and backslash/metacharacters are not permitted.

**elements_per_variable**

- Keywords Area
- variables
- discrete_state_set
- string
- elements_per_variable

Number of admissible elements for each set variable

**Specification**

Alias: `num_set_values`

Argument(s): INTEGERLIST

Default: equal distribution

**Description**

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

**elements**

- Keywords Area
- variables
- discrete_state_set
- string
6.4. VARIABLES

- elements

The permissible values for each discrete variable

**Specification**

**Alias:** set_values

**Argument(s):** STRINGLIST

**Description**

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the discrete variables page.

**initial_state**

- Keywords Area
- variables
- discrete_state_set
- string
- initial_state

Initial values for the state variables

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** middle set value, or rounded down

**Description**

The initial_state specifications provide the initial values for the state variables.

This is an optional keyword. If it is not specified, the initial state will be inferred from the other keywords that define the state variable.

Defaults are:

- Continuous state variables - use the midpoint of the bounds
- Set variables - use the value with the index closest to the middle of the set
- Range variables - use the midpoint of the range
descriptors

- Keywords Area
- variables
- discrete_state_set
- string
- descriptors

Labels for the variables

Specification

Alias: none
Argument(s): STRINGLIST
Default: dssv_{i}

Description

The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

real

- Keywords Area
- variables
- discrete_state_set
- real

Discrete state variables, each defined by a set of permissible real numbers

Topics

This keyword is related to the topics:

- discrete_variables
- state_variables

Specification

Alias: none
Argument(s): INTEGER
Default: no discrete state set real variables
### 6.4. VARIABLES

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>elements_per_variable</td>
<td>elements</td>
<td>Number of admissible elements for each set variable</td>
</tr>
<tr>
<td>Required</td>
<td>elements</td>
<td>categorical</td>
<td>The permissible values for each discrete variable</td>
</tr>
<tr>
<td>Optional</td>
<td>initial_state</td>
<td>descriptors</td>
<td>Whether the set-valued variables are categorical or relaxable</td>
</tr>
<tr>
<td>Optional</td>
<td>descriptors</td>
<td></td>
<td>Initial values for the state variables</td>
</tr>
</tbody>
</table>

**Description**

Discrete state variables defined by a set of permissible real numbers.

The details of how to specify this discrete variable are located on the `discrete_variables` page.

See the `state_variables` page for details on the behavior of state variables.

**elements_per_variable**

- Keywords Area
- variables
- discrete_state_set
- real
- elements_per_variable

Number of admissible elements for each set variable

**Specification**

**Alias:** num_set_values

**Argument(s):** INTEGERLIST

**Default:** equal distribution
CHAPTER 6. KEYWORDS AREA

Description
Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type
(integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If
not specified, equal apportionment of elements among variables is assumed, and the number of elements must be
evenly divisible by the number of variables.

elements

- Keywords Area
- variables
- discrete_state_set
- real
- elements

The permissible values for each discrete variable

Specification
Alias: set_values
Argument(s): REALLIST

Description
Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on
the discrete_variables page.

categorical

- Keywords Area
- variables
- discrete_state_set
- real
- categorical

Whether the set-valued variables are categorical or relaxable

Specification
Alias: none
Argument(s): STRINGLIST

Description
A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are
strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on
any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes',
'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*
Examples

Discrete design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

discrete_design_set
default: integer 1
elements 2 4 7
descriptor 'rotor_blades'
categorical 'no'

initial_state

- Keywords Area
- variables
- discrete_state_set
- real
- initial_state

Initial values for the state variables

Specification

Alias: none

Argument(s): REALLIST

Default: middle set value, or rounded down

Description

The initial_state specifications provide the initial values for the state variables. This is an optional keyword. If it is not specified, the initial state will be inferred from the other keywords that define the state variable.

Defaults are:

- Continuous state variables - use the midpoint of the bounds
- Set variables - use the value with the index closest to the middle of the set
- Range variables - use the midpoint of the range

descriptors

- Keywords Area
- variables
- discrete_state_set
- real
- descriptors

Labels for the variables
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
Argument(s): STRINGLIST
Default: dssrv_{i}

Description
The optional variables labels specification descriptors is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a root string plus a numeric identifier.

6.5 interface

- Keywords Area
- interface

Specifies how function evaluations will be performed in order to map the variables into the responses.

Topics
This keyword is related to the topics:

- block

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>id_interface</td>
<td>Name the interface block; helpful when there are multiple</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>algebraic_-mappings</td>
<td>Use AMPL to define algebraic input-output mappings</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>analysis_drivers</td>
<td>Define how Dakota should run a function evaluation</td>
</tr>
</tbody>
</table>
Description

The interface section in a Dakota input file specifies how function evaluations will be performed in order to map the variables into the responses.

In this context, a "function evaluation" is the series of operations that takes the variables and computes the responses. This can be comprised of one or many codes, scripts, and glue, which are generically referred to as "analysis drivers".

The optional asynchronous flag specifies use of asynchronous protocols (i.e., background system calls, nonblocking forks, POSIX threads) when evaluations or analyses are invoked. The evaluation_concurrency and analysis_concurrency specifications serve a dual purpose:

- when running Dakota on a single processor in asynchronous mode, the default concurrency of evaluations and analyses is all concurrency that is available. The evaluation_concurrency and analysis_concurrency specifications can be used to limit this concurrency in order to avoid machine overload or usage policy violation.

- when running Dakota on multiple processors in message passing mode, the default concurrency of evaluations and analyses on each of the servers is one (i.e., the parallelism is exclusively that of the message
passing). With the evaluation_concurrency and analysis_concurrency specifications, a hybrid parallelism can be selected through combination of message passing parallelism with asynchronous parallelism on each server.

If Dakota’s automatic parallel configuration is undesirable for some reason, the user can specify overrides that enforce a desired number of partitions, a size for the partitions, and/or a desired scheduling configuration at the evaluation and analysis parallelism levels. The optional evaluation_servers and analysis_servers specifications support user overrides of the automatic parallel configuration for the number of evaluation servers and the number of analysis servers, and the optional processors_per_evaluation specification supports user overrides for the size of processor allocations for evaluation servers (Note: see direct for the processors_per_analysis specification supported for direct interfaces). Similarly, the optional evaluation_scheduling and analysis_scheduling specifications can be used to override the automatic parallel configuration at the evaluation and analysis parallelism levels to use either a dedicated master or a peer partition. In addition, the evaluation parallelism level supports an override for the scheduling algorithm used within a peer partition; this can be either dynamic or static scheduling (default configuration of a peer partition employs a dynamic scheduler when it can be supported; i.e., when the peer 1 local scheduling can be asynchronous). The Parallel-Library class and the Parallel Computing chapter of the Users Manual[4] provide additional details on parallel configurations.

When performing asynchronous local evaluations, the local_evaluation_scheduling keyword controls how new evaluation jobs are dispatched when one completes. If the local_evaluation_scheduling is specified as dynamic (the default), each completed evaluation will be replaced by the next in the local evaluation queue. If local_evaluation_scheduling is specified as static, each completed evaluation will be replaced by an evaluation number that is congruent modulo the evaluation_concurrency. This is helpful for relative node scheduling as described in Dakota/examples/parallelism. For example, assuming only asynchronous local concurrency (no MPI), if the local concurrency is 6 and job 2 completes, it will be replaced with job 8. For the case of hybrid parallelism, static local scheduling results in evaluation replacements that are modulo the total capacity, defined as the product of the evaluation concurrency and the number of evaluation servers. Both of these cases can result in idle processors if runtimes are non-uniform, so the default dynamic scheduling is preferred when relative node scheduling is not required.

Theory

Function evaluations are performed using either interfaces to simulation codes, algebraic mappings, or a combination of the two.

When employing mappings with simulation codes, the interface invokes the simulation using either forks, direct function invocations, or computational grid invocations.

- In the fork case, Dakota will treat the simulation as a black-box and communication between Dakota and the simulation occurs through parameter and result files. This is the most common case.

- In the direct function case, the simulation is internal to Dakota and communication occurs through the function parameter list. The direct case can involve linked simulation codes or test functions which are compiled into the Dakota executable. The test functions allow for rapid testing of algorithms without process creation overhead or engineering simulation expense.

- The grid case is experimental and under development, but is intended to support simulations which are external to Dakota and geographically distributed.

When employing algebraic mappings, the AMPL solver library[29] is used to evaluate a directed acyclic graph (DAG) specification from a separate stub.nl file. Separate stub.col and stub.row files are also required to declare the string identifiers of the subset of inputs and outputs, respectively, that will be used in the algebraic mappings.
6.5. INTERFACE

6.5.1  id_interface

- Keywords Area
- interface
- id_interface

Name the interface block; helpful when there are multiple

Topics

This keyword is related to the topics:

- block_identifier

Specification

Alias: none

Argument(s): STRING

Default: use of last interface parsed

Description

The optional set identifier specification uses the keyword id_interface to input a string for use in identifying a particular interface specification. A model can then identify the use of this interface by specifying the same string in its interface_pointer specification.

If the id_interface specification is omitted, a particular interface specification will be used by a model only if that model omits specifying a interface_pointer and if the interface set was the last set parsed (or is the only set parsed). In common practice, if only one interface set exists, then id_interface can be safely omitted from the interface specification and interface_pointer can be omitted from the model specification(s), since there is no potential for ambiguity in this case.

Examples

For example, a model whose specification contains interface_pointer = 'I1' will use an interface specification with id_interface = 'I1'.

6.5.2  algebraic_mappings

- Keywords Area
- interface
- algebraic_mappings

Use AMPL to define algebraic input-output mappings

Specification

Alias: none

Argument(s): STRING

Default: no algebraic mappings
Description

If desired, one can define algebraic input-output mappings using the AMPL code[26] and save these mappings in 3 files: stub.nl, stub.col, and stub.row, where stub is a particular root name describing a particular problem. These files names can be communicated to Dakota using the algebraic_mappings input. This string may either specify the stub.nl filename, or alternatively, just the stub itself.

Dakota then uses stub.col and stub.row to extract the input and output identifier strings and employs the AMPL solver library[29] to process the DAG specification in stub.nl. The variable and objective function names declared within AMPL should be a subset of the variable descriptors and response descriptors used by Dakota (see variables and descriptors). Ordering is not important, as Dakota will reorder data as needed.

6.5.3 analysis_drivers

- Keywords Area
- interface
- analysis_drivers

Define how Dakota should run a function evaluation

Specification

Alias: none

Argument(s): STRINGLIST

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>analysis_components</td>
<td>Dakota Keyword</td>
<td>Provide additional identifiers to analysis drivers. Run a pre-processing script before the analysis drivers</td>
</tr>
<tr>
<td>Optional</td>
<td>input_filter</td>
<td></td>
<td>Run a post-processing script after the analysis drivers</td>
</tr>
<tr>
<td>Optional</td>
<td>output_filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>Group 1</td>
<td>system</td>
<td>(Not recommended) Launch analysis drivers with a system call</td>
</tr>
</tbody>
</table>
### Description

The required `analysis_drivers` keyword provides the names of one or more executable analysis programs or scripts, a.k.a. "drivers" which comprise a function evaluation. The optional and required sub-keywords specify how Dakota will manage directories and files, and run the driver(s).

#### Types of Interfaces

Dakota has two recommended ways of running analysis drivers:

- as an external processes (`fork`), or
- using internal code to couple to the analysis driver (`direct`)

Other options are available for advanced users, and are not as well documented, supported, or tested:

- external processes (`system`)
- internal coupling (`python, matlab, scilab, grid`)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td>Launch analysis drivers using fork command</td>
</tr>
<tr>
<td>direct</td>
<td>Run analysis drivers that are linked-to or compiled-with Dakota</td>
</tr>
<tr>
<td>matlab</td>
<td>Run Matlab with a direct interface - requires special Dakota build</td>
</tr>
<tr>
<td>python</td>
<td>Run Python with a direct interface - requires special Dakota build</td>
</tr>
<tr>
<td>scilab</td>
<td>Run Scilab with a direct interface - requires special Dakota build</td>
</tr>
<tr>
<td>grid</td>
<td>Experimental capability</td>
</tr>
<tr>
<td>failure_capture</td>
<td>Determine how Dakota responds to analysis driver failure</td>
</tr>
<tr>
<td>deactivate</td>
<td>Deactivate Dakota features to simplify interface development, increase execution speed, or reduce memory and disk requirements</td>
</tr>
</tbody>
</table>
Use Cases
The internally coupled codes have few options because many of the details are already handled with the coupling. Their behavior is described in the direct keyword.

For external processes using the fork keyword,

A function evaluation may comprise:

1. A single analysis driver: Function evaluation, including all pre- and post-processing is contained entirely within a single script/executable.

2. A single analysis driver with filters: Function evaluation is explicitly split into pre-processing (performed by the input filter), analysis, and post-processing (by the output filter).

3. A single analysis driver with environment variables: Function evaluation is contained within one analysis driver, but it requires environment variables to be set before running.

4. Multiple analysis drivers: Drivers are run sequentially or concurrently (See the asynchronous keyword) and can have any of the above options as well.

For fork and system interfaces, the analysis driver list contains the names of one or more executable programs or scripts taking parameters files as input and producing results files as output. The first field in each analysis driver string must be an executable program or script for Dakota to spawn to perform the function evaluation.

Drivers support:

- One set of nested quotes, for arguments with spaces
- Dakota will define special environment variables DAKOTA_PARAMETERS_FILE and DAKOTA_RESULTS_FILE which can be used in the driver script.
- Variable definitions preceding the executable program or script, such as 'MY_VAR=2 run_analysis.sh' are no longer supported.

For details and examples see the Simulation Interface Components section of the Interfaces chapter of the User’s Manual; for details on the filters and environment variables, see the subsection on Syntax for Filter and Driver Strings.

Examples
Examples:

1. analysis_drivers = ‘run_simulation_part1.sh’ ‘run_simulation_part2.sh’

2. analysis_driver = ‘run_simulation.sh -option “option 1”’

3. analysis_driver = ‘simulation.exe -option value -dakota_params $DAKOTA_PARAMETERS_FILE -input sim.in -dakota_results_file $DAKOTA_RESULTS_FILE’

FAQ
Where will Dakota look for the analysis driver? Dakota will locate analysis_driver programs first in (or relative to) the present working directory ("."), the interface-analysis_drivers-fork-work_directory if used, otherwise the directory in which Dakota is started, then the directory from which Dakota is started, then using the system $PATH environment variable (Path% on Windows).

Where should the driver be located? When the driver is a script it is most commonly placed in the same directory as the Dakota input file. When using a work_directory, Dakota will also look for drivers in the specified working directory, so link_files or copy_files may specify the driver to get copied or linked into the work directory.
When executable programs are used as drivers, they are often elsewhere on the filesystem. These can be specified using absolute paths, or by prepending the PATH environment variable so Dakota finds them.

**What if Dakota fails to run my analysis_driver?** Prepend the absolute location of the driver to the PATH environment variable before running Dakota, or specify an absolute path to the driver in the Dakota input file.

```
analysis_components
  • Keywords Area
  • interface
  • analysis_drivers
  • analysis_components
```

Provide additional identifiers to analysis drivers.

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** no additional identifiers

**Description**

The optional `analysis_components` specification allows the user to provide additional identifiers (e.g., mesh file names) for use by the analysis drivers. This is particularly useful when the same analysis driver is to be reused multiple times for slightly different analyses. The specific content within the strings is open-ended and can involve whatever syntax is convenient for a particular analysis driver. The number of analysis components $n_c$ should be an integer multiple of the number of drivers $n_d$, and the first $n_c/n_d$ component strings will be passed to the first driver, etc.

```
input_filter
  • Keywords Area
  • interface
  • analysis_drivers
  • input_filter
```

Run a pre-processing script before the analysis drivers.

**Specification**

**Alias:** none

**Argument(s):** STRING

**Default:** no input filter
CHAPTER 6. KEYWORDS AREA

Description

The optional input_filter and output_filter specifications provide the names of separate pre- and post-processing programs or scripts which assist in mapping Dakota parameters files into analysis input files and mapping analysis output files into Dakota results files, respectively.

If there is only a single analysis driver, then it is usually most convenient to combine pre- and post-processing requirements into a single analysis driver script and omit the separate input and output filters. However, in the case of multiple analysis drivers, the input and output filters provide a convenient location for non-repeated pre- and post-processing requirements. That is, input and output filters are only executed once per function evaluation, regardless of the number of analysis drivers, which makes them convenient locations for data processing operations that are shared among the analysis drivers.

output_filter

- Keywords Area
- interface
- analysis_drivers
- output_filter

Run a post-processing script after the analysis drivers

Specification

Alias: none

Argument(s): STRING
Default: no output filter

Description

The optional input_filter and output_filter specifications provide the names of separate pre- and post-processing programs or scripts which assist in mapping Dakota parameters files into analysis input files and mapping analysis output files into Dakota results files, respectively.

If there is only a single analysis driver, then it is usually most convenient to combine pre- and post-processing requirements into a single analysis driver script and omit the separate input and output filters. However, in the case of multiple analysis drivers, the input and output filters provide a convenient location for non-repeated pre- and post-processing requirements. That is, input and output filters are only executed once per function evaluation, regardless of the number of analysis drivers, which makes them convenient locations for data processing operations that are shared among the analysis drivers.

system

- Keywords Area
- interface
- analysis_drivers
- system

(Not recommended) Launch analysis drivers with a system call
## Specification

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>parameters_file</td>
<td>Specify the name of the parameters file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>results_file</td>
<td>Specify the name of the results file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>allow_results</td>
<td>Change how Dakota deals with existing results files</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>verbatim</td>
<td>Specify the command Dakota uses to launch analysis driver(s) and filters</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>aprepro</td>
<td>Write parameters files in APREPRO syntax</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>labeled</td>
<td>Requires correct function value labels in results file</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>file_tag</td>
<td>Tag each parameters &amp; results file name with the function evaluation number</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>file_save</td>
<td>Keep the parameters &amp; results files after the analysis driver completes</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>work_directory</td>
<td>Perform each function evaluation in a separate working directory</td>
</tr>
</tbody>
</table>

## Description

The system call interface is included in Dakota for portability and backward compatibility. Users are strongly encouraged to use the **fork** interface if possible, reverting to system only when necessary. To enable the system call interface, replace the **fork** keyword with **system**. All other keywords have identical meanings to those for the fork interface.
See Also
These keywords may also be of interest:

- fork

parameters_file

- Keywords Area
- interface
- analysis_drivers
- system
- parameters_file

Specify the name of the parameters file

Specification
Alias: none
Argument(s): STRING
Default: Unix temp files

Description
The parameters file is used by Dakota to pass the parameter values to the analysis driver. The name of the file can be optionally specified using the parameters_file keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., /usr/tmp/aaaa08861).

results_file

- Keywords Area
- interface
- analysis_drivers
- system
- results_file

Specify the name of the results file

Specification
Alias: none
Argument(s): STRING
Default: Unix temp files
6.5. INTERFACE

Description

The results file must be written by the analysis driver. It is read by Dakota to determine the response values for each function evaluation.

The name of the file can be optionally specified using the \texttt{results\_file} keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., /usr/tmp/aaaa08861).

allow\_existing\_results

- Keywords Area
- interface
- analysis\_drivers
- system
- allow\_existing\_results

Change how Dakota deals with existing results files

Specification

Alias: none

Argument(s): none

Default: results files removed before each evaluation

Description

By default Dakota will remove existing results files before invoking the \texttt{analysis\_driver} to avoid problems created by stale files in the current directory. To override this behavior and not delete existing files, specify \texttt{allow\_existing\_results}.

verbatim

- Keywords Area
- interface
- analysis\_drivers
- system
- verbatim

Specify the command Dakota uses to launch analysis driver(s) and filters

Specification

Alias: none

Argument(s): none

Default: driver/filter invocation syntax augmented with file names
Description

The typical commands that Dakota uses to launch analysis drivers are:

```bash
> analysis_driver parameters_file_name results_file_name
```

Dakota will automatically arrange the executables and file names.
If the analysis driver requires a different syntax, the entire command can be specified as the analysis driver and the `verbatim` keyword will tell Dakota to use this as the command.

Note, this will not allow the use of `file_tag`, because the exact command must be specified.
For additional information on invocation syntax, see the Interfaces chapter of the Users Manual[4].

Examples

In the following example, the analysis_driver command is run without any edits from Dakota.

```bash
interface
  analysis_driver = "matlab -nodesktop -nojvm -r 'MatlabDriver_hardcoded_filenames; exit' "
  fork
    parameters_file 'params.in'
    results_file 'results.out'
  verbatim # this tells Dakota to fork the command exactly as written, instead of appending I/O filenames
```

The `-r` flag identifies the commands that will be run by `matlab`. The `matlab` script has the `parameters_file` and `results_file` names hardcoded, so no additional arguments are required.

**aprepro**

- **Keywords Area**
- **interface**
- **analysis_drivers**
- **system**
- **aprepro**

Write parameters files in APREPRO syntax

Topics

This keyword is related to the topics:

- **file_formats**

Specification

**Alias:** dprepro

**Argument(s):** none

**Default:** standard parameters file format
6.5. INTERFACE

Description

The format of data in the parameters files can be modified for direct usage with the APREPRO pre-processing tool [75] using the `aprepro` specification.

Without this keyword, the parameters file are written in DPrePro format. DPrePro is a utility included with Dakota, described in the Users Manual[4].

**labeled**

- Keywords Area
- interface
- analysis_drivers
- system
- labeled

Requires correct function value labels in results file

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: Function value labels optional

Description

The `labeled` keyword directs Dakota to enforce a stricter results file format and enables more detailed error reporting.

When the `labeled` keyword is used, function values in results files must be accompanied by their corresponding descriptors. If the user did not supply response descriptors in her Dakota input file, then Dakota auto-generated descriptors are expected.

Distinct error messages are emitted for function values that are out-of-order, repeated, or missing. Labels that appear without a function value and unexpected data are also reported as errors. Dakota attempts to report all errors in a results file, not just the first it encounters. After reporting results file errors, Dakota aborts.

Labels for analytic gradients and Hessians currently are not supported.

Although the `labeled` keyword is optional, its use is recommended to help catch and identify problems with results files. The User’s Manual contains further information about the results file format.

Default Behavior

By default, Dakota does not require labels for function values, and ignores them if they are present.
CHAPTER 6. KEYWORDS AREA

file_tag

- Keywords Area
- interface
- analysis_drivers
- system
- file_tag

Tag each parameters & results file name with the function evaluation number

Specification

Alias: none
Argument(s): none
Default: no tagging

Description

If this keyword is used, Dakota will append a period and the function evaluation number to the names of the parameter and results files.

For example, if the following is included in the interface section of the Dakota input:

```
parameters_file = params.in
results_file = results.out
file_tag
```

Then for the 3rd evaluation, Dakota will write `params.in.3`, and will expect `results.out.3` to be written by the analysis driver.

If this keyword is omitted, the default is no file tagging.

File tagging is most useful when multiple function evaluations are running simultaneously using files in a shared disk space. The analysis driver will be able to infer the function evaluation number from the file names. Note that when the file_save keyword is used, Dakota renames parameters and results files, giving them tags, after execution of the analysis driver if they otherwise would be overwritten by the next evaluation.

file_save

- Keywords Area
- interface
- analysis_drivers
- system
- file_save

Keep the parameters & results files after the analysis driver completes

Specification

Alias: none
Argument(s): none
Default: file cleanup
6.5. INTERFACE

**Description**

If `file_save` is used, Dakota will not delete the parameters and results files after the function evaluation is completed.

The default behavior is NOT to save these files.

If `file_tag` is not specified and the saved files would be overwritten by a future evaluation, Dakota renames them after the analysis driver has run by tagging them with the evaluation number.

File saving is most useful when debugging the data communication between Dakota and the simulation.

**work_directory**

- Keywords Area
- `interface`
- `analysis_drivers`
- `system`
- `work_directory`

Perform each function evaluation in a separate working directory

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** no work directory

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td><code>named</code></td>
<td>The base name of the work directory created by Dakota</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>directory_tag</code></td>
<td>Tag each work directory with the function evaluation number</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td><code>directory_save</code></td>
<td>Preserve the work directory after function evaluation completion</td>
</tr>
</tbody>
</table>
**Description**

When performing concurrent evaluations, it is typically necessary to cloister simulation input and output files in separate directories to avoid conflicts. When the `work_directory` feature is enabled, Dakota will create a directory for each evaluation, with optional tagging (`directory_tag`) and saving (`directory_save`), as with files, and execute the analysis driver from that working directory.

The directory may be named with a string, or left anonymous to use an automatically-generated directory in the system’s temporary file space, e.g., `/tmp/dakota_work_c93vb71z/`. The optional `link_files` and `copy_files` keywords specify files or directories which should appear in each working directory.

When using `work_directory`, the `analysis_drivers` may be given by an absolute path, located in (or relative to) the startup directory alongside the Dakota input file, in the list of template files linked or copied, or on the `$PATH` (Path% on Windows).

named

- Keywords Area
- interface
- `analysis_drivers`
- system
- `work_directory`
- named

The base name of the work directory created by Dakota

**Specification**

**Alias:** none  
**Argument(s):** STRING  
**Default:** workdir
6.5. INTERFACE

Description
The named keyword is followed by a string, indicating the name of the work directory created by Dakota. If relative, the work directory will be created relative to the directory from which Dakota is invoked.

If named is not used, the default work directory is a temporary directory with a system-generated name (e.g., /tmp/dakota_work_c93vb71z/).

See Also
These keywords may also be of interest:
- directory_tag
- directory_save

directory_tag
- Keywords Area
- interface
- analysis_drivers
- system
- work_directory
- directory_tag
Tag each work directory with the function evaluation number

Specification
Alias: dir_tag
Argument(s): none
Default: no work directory tagging

Description
If this keyword is used, Dakota will append a period and the function evaluation number to the work directory names.

If this keyword is omitted, the default is no tagging, and the same work directory will be used for ALL function evaluations. Tagging is most useful when multiple function evaluations are running simultaneously.

directory_save
- Keywords Area
- interface
- analysis_drivers
- system
- work_directory
- directory_save
Preserve the work directory after function evaluation completion
CHAPTER 6. KEYWORDS AREA

**Specification**

**Alias:** dir_save

**Argument(s):** none

**Default:** remove work directory

**Description**

By default, when a working directory is created by Dakota using the `work_directory` keyword, it is deleted after the evaluation is completed. The `directory.save` keyword will cause Dakota to leave (not delete) the directory.

```plaintext
link_files
```

- Keywords Area
- interface
- analysis_drivers
- system
- work_directory
- link_files

Paths to be linked into each working directory

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** no linked files

**Description**

Specifies the paths (files or directories) that will be symbolically linked from each working directory. Wildcards using * and ? are permitted. Linking is space-saving and useful for files not modified during the function evaluation. However, not all filesystems support linking, for example, support on Windows varies.

**Examples**

Specifying

```plaintext
link_files = 'aiminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```plaintext
workdir/siminput*.in     # links to each of rundir / siminput*.in
workdir/simdir1/         # whole directory simdir1 linked
workdir/*                # each entry in directory simdir2 linked
```
6.5. INTERFACE

**copy_files**

- Keywords Area
- interface
- analysis_drivers
- system
- work_directory
- copy_files

Files and directories to be copied into each working directory

**Specification**

*Alias:* none

*Argument(s):* STRINGLIST

*Default:* no copied files

**Description**

Specifies the files or directories that will be recursively copied into each working directory. Wildcards using * and ? are permitted.

**Examples**

Specifying

```
copy_files = 'siminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```
workdir/siminput*.in  # files rundir/siminput*.in copied
workdir/simdir1/     # whole directory simdir1 recursively copied
workdir/*           # contents of directory simdir2 recursively copied
```

where rundir is the directory in which Dakota was started.

**replace**

- Keywords Area
- interface
- analysis_drivers
- system
- work_directory
- replace

Overwrite existing files within a work directory
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
Argument(s): none
Default: do not overwrite files

Description

By default, Dakota will not overwrite any existing files in a work directory. The replace keyword changes this behavior to force overwriting.

fork

- Keywords Area
- interface
- analysis_drivers
- fork

Launch analysis drivers using fork command

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>parameters_file</td>
<td></td>
<td>Specify the name of the parameters file</td>
</tr>
<tr>
<td>Optional</td>
<td>results_file</td>
<td></td>
<td>Specify the name of the results file</td>
</tr>
<tr>
<td>Optional</td>
<td>allow_existing_results</td>
<td></td>
<td>Change how Dakota deals with existing results files</td>
</tr>
<tr>
<td>Optional</td>
<td>verbatim</td>
<td></td>
<td>Specify the command Dakota uses to launch analysis driver(s) and filters</td>
</tr>
</tbody>
</table>
6.5. INTERFACE

| Optional                  | aprepro                                | Write parameters files in APREPRO syntax
|                          | labeled                                 | Requires correct function value labels in results file
|                          | file_tag                                | Tag each parameters & results file name with the function evaluation number
|                          | file_save                               | Keep the parameters & results files after the analysis driver completes
|                          | work_directory                          | Perform each function evaluation in a separate working directory

Description

The fork interface is the most common means by which Dakota launches a separate application analysis process. The fork interface is recommended over system for most analysis drivers that are external to Dakota (i.e. not using the direct interface).

As explained in the Users Manual, the parameters and results file names are passed on the command line to the analysis driver(s). If input/output filters are specified, they will be run before/after the analysis drivers. The verbatim keyword is used to modify the default driver/filter commands.

For additional information on invocation syntax, see the Interfaces chapter of the Users Manual[4].

Examples

```plaintext
interface
  analysis_drivers = ‘rosenbrock’
  fork
    parameters_file = ‘params.in’
    results_file   = ‘results.out’
    file_tag
    file_save
```

parameters_file

- Keywords Area
- interface
- analysis_drivers
- fork
- parameters_file
Specify the name of the parameters file

**Specification**

Alias: none  
Argument(s): STRING  
Default: Unix temp files

**Description**

The parameters file is used by Dakota to pass the parameter values to the analysis driver. The name of the file can be optionally specified using the `parameters_file` keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., `/usr/tmp/aaaa08861`).

results_file

- Keywords Area
- interface
- analysis_drivers
- fork
- results_file

Specify the name of the results file

**Specification**

Alias: none  
Argument(s): STRING  
Default: Unix temp files

**Description**

The results file must be written by the analysis driver. It is read by Dakota to determine the response values for each function evaluation.

The name of the file can be optionally specified using the `results_file` keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., `/usr/tmp/aaaa08861`).

allow_existing_results

- Keywords Area
- interface
- analysis_drivers
- fork
- allow_existing_results

Change how Dakota deals with existing results files
6.5. INTERFACE

**Specification**

Alias: none
- **Argument(s):** none
- **Default:** results files removed before each evaluation

**Description**

By default Dakota will remove existing results files before invoking the `analysis_driver` to avoid problems created by stale files in the current directory. To override this behavior and not delete existing files, specify `allow-existing-results`.

**verbatim**
- **Keywords Area**
- **interface**
- **analysis_drivers**
- **fork**
- **verbatim**
  
  Specify the command Dakota uses to launch analysis driver(s) and filters

**Specification**

Alias: none
- **Argument(s):** none
- **Default:** driver/filter invocation syntax augmented with file names

**Description**

The typical commands that Dakota uses to launch analysis drivers are:

```plaintext
> analysis_driver parameters_file_name results_file_name
```

Dakota will automatically arrange the executables and file names.

If the analysis driver requires a different syntax, the entire command can be specified as the analysis driver and the `verbatim` keyword will tell Dakota to use this as the command.

Note, this will not allow the use of `file_tag`, because the exact command must be specified.

For additional information on invocation syntax, see the Interfaces chapter of the Users Manual[4].

**Examples**

In the following example, the `analysis_driver` command is run without any edits from Dakota.

```plaintext
interface
analysis_driver = "matlab -nodesktop -nojvm -r 'MatlabDriver_hardcoded_filenames; exit' "
fork
parameters_file ‘params.in’
results_file ‘results.out’
verbatim // this tells Dakota to fork the command exactly as written, instead of appending I/O filenames
```

The `-r` flag identifies the commands that will be run by `matlab`. The `Matlab` script has the `parameters_file` and `results_file` names hardcoded, so no additional arguments are required.
aprepro

- Keywords Area
- interface
- analysis_drivers
- fork
- aprepro

Write parameters files in APREPRO syntax

**Topics**

This keyword is related to the topics:

- file_formats

**Specification**

Alias: dprepro

*Argument(s):* none

*Default:* standard parameters file format

**Description**

The format of data in the parameters files can be modified for direct usage with the APREPRO pre-processing tool [75] using the aprepro specification.

Without this keyword, the parameters file are written in DPrePro format. DPrePro is a utility included with Dakota, described in the Users Manual[4].

labeled

- Keywords Area
- interface
- analysis_drivers
- fork
- labeled

Requires correct function value labels in results file

**Topics**

This keyword is related to the topics:

- file_formats
6.5. INTERFACE

Specification

Alias: none

Argument(s): none

Default: Function value labels optional

Description

The labeled keyword directs Dakota to enforce a stricter results file format and enables more detailed error reporting.

When the labeled keyword is used, function values in results files must be accompanied by their corresponding descriptors. If the user did not supply response descriptors in her Dakota input file, then Dakota auto-generated descriptors are expected.

Distinct error messages are emitted for function values that are out-of-order, repeated, or missing. Labels that appear without a function value and unexpected data are also reported as errors. Dakota attempts to report all errors in a results file, not just the first it encounters. After reporting results file errors, Dakota aborts.

Labels for analytic gradients and Hessians currently are not supported.

Although the labeled keyword is optional, its use is recommended to help catch and identify problems with results files. The User’s Manual contains further information about the results file format.

Default Behavior

By default, Dakota does not require labels for function values, and ignores them if they are present.

file_tag

- Keywords Area
- interface
- analysis_drivers
- fork
- file_tag

Tag each parameters & results file name with the function evaluation number

Specification

Alias: none

Argument(s): none

Default: no tagging

Description

If this keyword is used, Dakota will append a period and the function evaluation number to the names of the parameter and results files.

For example, if the following is included in the interface section of the Dakota input:

```plaintext
parameters_file = params.in
results_file = results.out
file_tag
```
Then for the 3rd evaluation, Dakota will write `params.in.3`, and will expect `results.out.3` to be written by the analysis driver.

If this keyword is omitted, the default is no file tagging.

File tagging is most useful when multiple function evaluations are running simultaneously using files in a shared disk space. The analysis driver will be able to infer the function evaluation number from the file names. Note that when the `file_save` keyword is used, Dakota renames parameters and results files, giving them tags, after execution of the analysis driver if they otherwise would be overwritten by the next evaluation.

**file_save**

- **Keywords Area**
- **interface**
- **analysis_drivers**
- **fork**
- **file_save**

Keep the parameters & results files after the analysis driver completes

**Specification**

*Alias*: none  
*Argument(s)*: none  
*Default*: file cleanup

**Description**

If `file_save` is used, Dakota will not delete the parameters and results files after the function evaluation is completed.

The default behavior is **NOT** to save these files.

If `file_tag` is not specified and the saved files would be overwritten by a future evaluation, Dakota renames them after the analysis driver has run by tagging them with the evaluation number.

File saving is most useful when debugging the data communication between Dakota and the simulation.

**work_directory**

- **Keywords Area**
- **interface**
- **analysis_drivers**
- **fork**
- **work_directory**

Perform each function evaluation in a separate working directory

**Specification**

*Alias*: none  
*Argument(s)*: none  
*Default*: no work directory
### Description

When performing concurrent evaluations, it is typically necessary to cloister simulation input and output files in separate directories to avoid conflicts. When the `work_directory` feature is enabled, Dakota will create a directory for each evaluation, with optional tagging (`directory_tag`) and saving (`directory_save`), as with files, and execute the analysis driver from that working directory.

The directory may be named with a string, or left anonymous to use an automatically-generated directory in the system’s temporary file space, e.g., `/tmp/dakota_work_c93vb71z/`. The optional `link_files` and `copy_files` keywords specify files or directories which should appear in each working directory.

When using `work_directory`, the `analysis_drivers` may be given by an absolute path, located in (or relative to) the startup directory alongside the Dakota input file, in the list of template files linked or copied, or on the `$PATH` (PATH% on Windows).

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>named</td>
<td>The base name of the work directory created by Dakota</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>directory_tag</td>
<td>Tag each work directory with the function evaluation number</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>directory_save</td>
<td>Preserve the work directory after function evaluation completion</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>link_files</td>
<td>Paths to be linked into each working directory</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>copy_files</td>
<td>Files and directories to be copied into each working directory</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>replace</td>
<td>Overwrite existing files within a work directory</td>
</tr>
</tbody>
</table>

- Keywords Area
  - interface
  - analysis_drivers
  - fork
• work_directory

• named

The base name of the work directory created by Dakota

**Specification**

**Alias:** none  
**Argument(s):** STRING  
**Default:** workdir

**Description**

The `named` keyword is followed by a string, indicating the name of the work directory created by Dakota. If relative, the work directory will be created relative to the directory from which Dakota is invoked.

If `named` is not used, the default work directory is a temporary directory with a system-generated name (e.g., `/tmp/dakota_work_c93vb71z/`).

**See Also**

These keywords may also be of interest:

• directory_tag  
• directory_save

**directory_tag**

• Keywords Area  
• interface  
• analysis_drivers  
• fork  
• work_directory  
• directory_tag

Tag each work directory with the function evaluation number

**Specification**

**Alias:** dir_tag  
**Argument(s):** none  
**Default:** no work directory tagging

**Description**

If this keyword is used, Dakota will append a period and the function evaluation number to the work directory names.

If this keyword is omitted, the default is no tagging, and the same work directory will be used for ALL function evaluations. Tagging is most useful when multiple function evaluations are running simultaneously.
directory_save
- Keywords Area
- interface
- analysis_drivers
- fork
- work_directory
- directory_save

Preserve the work directory after function evaluation completion

**Specification**

**Alias:** dir_save

**Argument(s):** none

**Default:** remove work directory

**Description**

By default, when a working directory is created by Dakota using the `work_directory` keyword, it is deleted after the evaluation is completed. The `directory_save` keyword will cause Dakota to leave (not delete) the directory.

link_files
- Keywords Area
- interface
- analysis_drivers
- fork
- work_directory
- link_files

Paths to be linked into each working directory

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** no linked files

**Description**

Specifies the paths (files or directories) that will be symbolically linked from each working directory. Wildcards using * and ? are permitted. Linking is space-saving and useful for files not modified during the function evaluation. However, not all filesystems support linking, for example, support on Windows varies.
**Examples**

Specifying

```plaintext
link_files = 'siminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```plaintext
workdir/siminput*.in  # links to each of rundir / siminput*.in
workdir/simdir1/      # whole directory simdir1 linked
workdir/*             # each entry in directory simdir2 linked
```

**copy_files**

- **Keywords Area**
- **interface**
- **analysis_drivers**
- **fork**
- **work_directory**
- **copy_files**

Files and directories to be copied into each working directory

**Specification**

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** no copied files

**Description**

Specifies the files or directories that will be recursively copied into each working directory. Wildcards using * and ? are permitted.

**Examples**

Specifying

```plaintext
copy_files = 'siminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```plaintext
workdir/siminput*.in  # files rundir/siminput*.in copied
workdir/simdir1/      # whole directory simdir1 recursively copied
workdir/*             # contents of directory simdir2 recursively copied
```

where rundir is the directory in which Dakota was started.
replace

- Keywords Area
- interface
- analysis_drivers
- fork
- work_directory
- replace

Overwrite existing files within a work directory

**Specification**

**Alias:** none  
**Argument(s):** none  
**Default:** do not overwrite files

**Description**

By default, Dakota will not overwrite any existing files in a work directory. The `replace` keyword changes this behavior to force overwriting.

**direct**

- Keywords Area
- interface
- analysis_drivers
- direct

Run analysis drivers that are linked-to or compiled-with Dakota

**Specification**

**Alias:** none  
**Argument(s):** none

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional   | Group          |                | Description    |
|            |                | processors_per- | Specify the    |
|            |                | analysis        | number of      |
|            |                |                 | processors     |
|            |                |                 | per analysis   |
|            |                |                 | when Dakota    |
|            |                |                 | is run in      |
|            |                |                 | parallel       |
CHAPTER 6. KEYWORDS AREA

Description

The primary use of the direct interface is to invoke internal test functions that perform parameter to response mappings for simple functions as inexpensively as possible. These problems are compiled directly into the Dakota executable as part of the direct function interface class and are used for algorithm testing.

Dakota also supports direct interfaces to a few select simulation codes. One example is ModelCenter, a commercial simulation management framework from Phoenix Integration. To utilize this interface, a user must first define the simulation specifics within a ModelCenter session and then save these definitions to a ModelCenter configuration file. The analysis_components specification provides the means to communicate this configuration file to Dakota’s ModelCenter interface.

Examples

The rosenbrock function is available as an executable, which can be launched with fork, and is also compiled with Dakota. The internal version can be launched with:

```
interface
  analysis_drivers = 'rosenbrock'
  direct
```

Producers_per_analysis

- Keywords Area
- interface
- analysis_drivers
- direct
- processors_per_analysis

Specify the number of processors per analysis when Dakota is run in parallel.

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none
Argument(s): INTEGER
Default: no override of auto configure

Description

For direct function interfaces, processors_per_analysis is an additional optional setting within the required group which can be used to specify multiprocessor analysis partitions. As with the evaluation_servers, analysis_servers, evaluation_self_scheduling, evaluation_static_scheduling, analysis_self_scheduling, and analysis_static_scheduling specifications, processors_per_analysis provides a means for the user to override the automatic parallel configuration (refer to Parallel-Library and the Parallel Computing chapter of the Users Manual [4]) for the number of processors used for each analysis partition. Note that if both analysis_servers and processors_per_analysis are specified and they are not in agreement, then analysis_servers takes precedence.
6.5. INTERFACE

matlab

- Keywords Area
- interface
- analysis_drivers
- matlab

Run Matlab with a direct interface - requires special Dakota build

Specification

Alias: none
Argument(s): none

Description

Dakota supports library-linked interfaces to Matlab, Scilab, and Python scientific computation software, but they must be explicitly enabled when compiling Dakota from source. First consult the Users Manual[4] for discussion and examples.

Contact the Dakota users mailing list for assistance building and using Dakota with these interfaces.

In all these interfaces, the analysis_driver is used to specify a Matlab, Scilab, or Python file which implements the parameter to response mapping.

python

- Keywords Area
- interface
- analysis_drivers
- python

Run Python with a direct interface - requires special Dakota build

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>numpy</td>
<td>Enable the use of numpy in Dakota’s Python interface</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

Dakota supports library-linked interfaces to Matlab, Scilab, and Python scientific computation software, but they must be explicitly enabled when compiling Dakota from source. First consult the Users Manual[4] for discussion and examples.

Contact the Dakota users mailing list for assistance building and using Dakota with these interfaces.

In all these interfaces, the analysis_driver is used to specify a Matlab, Scilab, or Python file which implements the parameter to response mapping.

numpy

- Keywords Area
- interface
- analysis_drivers
- python
- numpy

Enable the use of numpy in Dakota’s Python interface

Specification

Alias: none

Argument(s): none

Default: Python list dataflow

Description

When the numpy keyword is used, Dakota expects responses in the form of a Python dictionary of numpy arrays. See the example in examples/linked_interfaces/Python.

scilab

- Keywords Area
- interface
- analysis_drivers
- scilab

Run Scilab with a direct interface - requires special Dakota build

Specification

Alias: none

Argument(s): none
Description
Dakota supports library-linked interfaces to Matlab, Scilab, and Python scientific computation software, but they must be explicitly enabled when compiling Dakota from source. First consult the Users Manual[4] for discussion and examples.

Contact the Dakota users mailing list for assistance building and using Dakota with these interfaces.

In all these interfaces, the `analysis_driver` is used to specify a Matlab, Scilab, or Python file which implements the parameter to response mapping.

grid

- Keywords Area
- interface
- analysis_drivers
- grid

Experimental capability

Specification
Alias: none
Argument(s): none

Description
For grid interfaces, no additional specifications are used at this time.

This capability has been used for interfaces with IDEA and JAVASpaces in the past and is currently a placeholder for future work with Condor and/or Globus services. It is not currently operational.

failure_capture

- Keywords Area
- interface
- analysis_drivers
- failure_capture

Determine how Dakota responds to analysis driver failure

Specification
Alias: none
Argument(s): none
Default: abort
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required (Choose One)</td>
<td>failure mitigation (Group 1)</td>
<td>abort</td>
<td>(Default) Abort the Dakota job</td>
</tr>
<tr>
<td>retry</td>
<td>Rerun failed analyses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recover</td>
<td>Substitute dummy values for the responses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>continuation</td>
<td>Cause Dakota to step toward the failed &quot;target&quot; simulation from a nearby successful &quot;source&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

Dakota can deal with analysis failure in a few ways.

The first step is that Dakota must detect analysis failure. Importantly, Dakota always expects a results file to be written by the analysis driver, even when a failure has occurred. If the file does not exist when the analysis driver exits, a Dakota error results, causing Dakota itself to terminate. The analysis driver communicates an analysis failure to Dakota by writing a results file beginning with the word "fail", which is not case sensitive. Everything after "fail" is ignored.

Once Dakota detects analysis failure, the failure can be mitigated in four ways:

- **abort** (the default)
- **retry**
- **recover**
- **continuation**


**abort**

- **Keywords Area**
- **interface**
- **analysis_drivers**
- **failure_capture**
- **abort**

(Defaults) Abort the Dakota job

**Specification**

Alias: none

**Argument(s):** none
6.5. INTERFACE

Description

abort will stop the Dakota job, as well as any other running analysis drivers.

retry

• Keywords Area
• interface
• analysis_drivers
• failure_capture
• retry

Rerun failed analyses

Specification

Alias: none
Argument(s): INTEGER

Description

The retry selection supports an integer input for specifying a limit on retries

recover

• Keywords Area
• interface
• analysis_drivers
• failure_capture
• recover

Substitute dummy values for the responses

Specification

Alias: none
Argument(s): REALLIST

Description

The recover selection supports a list of reals for specifying the dummy function values (only zeroth order information is supported) to use for the failed function evaluation.
CHAPTER 6. KEYWORDS AREA

continuation

- Keywords Area
- interface
- analysis_drivers
- failure_capture
- continuation

Cause Dakota to step toward the failed "target" simulation from a nearby successful "source"

Specification

Alias: none

Argument(s): none

Description

When failure_capture continuation is enabled and an evaluation fails, then Dakota will attempt to march incrementally from a previous good evaluation (the "source") toward the failing one (the "target"). Further details about the algorithm employed by Dakota are supplied in the User's Manual [4].

deactivate

- Keywords Area
- interface
- analysis_drivers
- deactivate

Deactivate Dakota features to simplify interface development, increase execution speed, or reduce memory and disk requirements

Specification

Alias: none

Argument(s): none

Default: Active set vector control, function evaluation cache, and restart file features are active

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>active_set_vector</td>
<td>Deactivate the Active Set Vector</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>evaluation_cache</td>
<td>Do not retain function evaluation history in memory</td>
</tr>
</tbody>
</table>
### Description

The optional `deactivate` specification block includes three features which a user may deactivate in order to simplify interface development, increase execution speed, and/or reduce memory and disk requirements:

- **Active set vector (ASV) control**: Deactivation of this feature using a `deactivate active_set_vector` specification allows the user to turn off any variability in ASV values so that active set logic can be omitted in the user’s simulation interface. This option trades some simplicity for simplicity in interface development. The default behavior is to request the minimum amount of data required by an algorithm at any given time, which implies that the ASV values may vary from one function evaluation to the next. Since the user’s interface must return the data set requested by the ASV values, this interface must contain additional logic to account for any variations in ASV content. Deactivating this ASV control causes Dakota to always request a “full” data set (the full function, gradient, and Hessian data that is available from the interface as specified in the responses specification) on each function evaluation. For example, if ASV control has been deactivated and the responses section specifies four response functions, analytic gradients, and no Hessians, then the ASV on every function evaluation will be `{3 3 3 3}`, regardless of what subset of this data is currently needed. While wasteful of computations in many instances, this simplifies the interface and allows the user to return the same data set on every evaluation. Conversely, if ASV control is active (the default behavior), then the ASV requests in this example might vary from `{1 1 1 1}` to `{2 0 0 2}`, etc., according to the specific data needed on a particular function evaluation. This will require the user’s interface to read the ASV requests and perform the appropriate logic in conditionally returning only the data requested. In general, the default ASV behavior is recommended for the sake of computational efficiency, unless interface development time is a critical concern. Note that in both cases, the data returned to Dakota from the user’s interface must match the ASV passed in, or else a response recovery error will result. However, when the ASV control is deactivated, the ASV values are invariant and need not be checked on every evaluation. **Note**: Deactivating the ASV control can have a positive effect on load balancing for parallel Dakota executions. Thus, there is significant overlap in this ASV control option with speculative gradients. There is also overlap with the mode override approach used with certain optimizers to combine individual value, gradient, and Hessian requests.

- **Function evaluation cache**: Deactivation of this feature using a `deactivate evaluation_cache` specification allows the user to avoid retention of the complete function evaluation history in memory. This can be important for reducing memory requirements in large-scale applications (i.e., applications with a large number of variables or response functions) and for eliminating the overhead of searching for duplicates within the function evaluation cache prior to each new function evaluation (e.g., for improving speed in problems with 1000’s of inexpensive function evaluations or for eliminating overhead when performing timing studies). However, the downside is that unnecessary computations may be performed since duplication in function evaluation requests may not be detected. For this reason, this option is not recommended when function evaluations are costly. **Note**: Duplication detection within Dakota can be deactivated, but duplication detection features within specific optimizers may still be active.

- **Strict Cache Equality**: By default, Dakota’s evaluation cache and restart capabilities are based on strict binary equality. This provides a performance advantage, as it permits a hash-based data structure to be used

<table>
<thead>
<tr>
<th>Optional</th>
<th>strict_cache_equality</th>
<th>Do not require strict cache equality when finding duplicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>restart_file</td>
<td>Deactivate writing to the restart file</td>
</tr>
</tbody>
</table>
to search the evaluation cache. However, deactivating strict equality may prevent cache misses, which can occur when attempting to use a restart file on a machine different from the one on which it was generated.

- **Restart file**: deactivation of this feature using a `deactivate restart_file` specification allows the user to eliminate the output of each new function evaluation to the binary restart file. This can increase speed and reduce disk storage requirements, but at the expense of a loss in the ability to recover and continue a run that terminates prematurely (e.g., due to a system crash or network problem). This option is not recommended when function evaluations are costly or prone to failure. Please note that using the `deactivate restart_file` specification will result in a zero length restart file with the default name `dakota.rst`.

These three features may be deactivated independently and concurrently.

**active_set_vector**

- **Keywords Area**
- **interface**
- **analysis_drivers**
- **deactivate**
- **active_set_vector**

Deactivate the Active Set Vector

**Specification**

Alias: none

*Argument(s):* none

**Description**

Described on parent page

**evaluation_cache**

- **Keywords Area**
- **interface**
- **analysis_drivers**
- **deactivate**
- **evaluation_cache**

Do not retain function evaluation history in memory

**Specification**

Alias: none

*Argument(s):* none
6.5. INTERFACE

Description
Described on parent page

strict_cache_equality

- Keywords Area
- interface
- analysis_drivers
- deactivate
- strict_cache_equality

Do not require strict cache equality when finding duplicates

Specification
Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>cache_tolerance</td>
<td>Specify tolerance when identifying duplicate function evaluations</td>
</tr>
</tbody>
</table>

Description
Described on parent page

cache_tolerance

- Keywords Area
- interface
- analysis_drivers
- deactivate
- strict_cache_equality
- cache_tolerance

Specify tolerance when identifying duplicate function evaluations

Specification
Alias: none
Argument(s): REAL
Description
Described on parent page

**restart_file**
- Keywords Area
- interface
- analysis,drivers
- deactivate
- restart_file

Deactivate writing to the restart file

**Specification**
Alias: none
Argument(s): none

Description
Described on parent page

6.5.4 **asynchronous**
- Keywords Area
- interface
- asynchronous

Specify analysis driver concurrency, when Dakota is run in serial

**Topics**
This keyword is related to the topics:
- concurrency_and_parallelism

**Specification**
Alias: none
Argument(s): none
Default: synchronous interface usage

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Description

The optional asynchronous keyword specifies use of asynchronous protocols (i.e., background system calls, nonblocking forks, POSIX threads) when evaluations or analyses are invoked. The `evaluation_concurrency` and `analysis_concurrency` specifications serve a dual purpose:

- when running Dakota on a single processor in asynchronous mode, the default concurrency of evaluations and analyses is all concurrency that is available. The `evaluation_concurrency` and `analysis_concurrency` specifications can be used to limit this concurrency in order to avoid machine overload or usage policy violation.

- when running Dakota on multiple processors in message passing mode, the default concurrency of evaluations and analyses on each of the servers is one (i.e., the parallelism is exclusively that of the message passing). With the `evaluation_concurrency` and `analysis_concurrency` specifications, a hybrid parallelism can be selected through combination of message passing parallelism with asynchronous parallelism on each server.

#### evaluation_concurrency

- **Keywords Area**
- **interface**
- **asynchronous**
- **evaluation_concurrency**

Determine how many concurrent evaluations Dakota will schedule.

### Topics

This keyword is related to the topics:

- **concurrency_and_parallelism**

### Specification

**Alias:** none

**Argument(s):** INTEGER

**Default:** local: unlimited concurrency, hybrid: no concurrency
CHAPTER 6. KEYWORDS AREA

**Description**

When asynchronous execution is enabled, the default behavior is to launch all available evaluations simultaneously. The `evaluation_concurrency` keyword can be used to limit the number of concurrent evaluations.

**local_evaluation_scheduling**

- **Keywords Area**
- **interface**
- **asynchronous**
- **local_evaluation_scheduling**

  Control how local asynchronous jobs are scheduled

**Topics**

This keyword is related to the topics:

- `concurrency_and_parallelism`

**Specification**

**Alias**: none

**Argument(s)**: none

**Default**: dynamic

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>dynamic</strong></td>
<td>Dynamic local scheduling (sequential)</td>
</tr>
</tbody>
</table>

**Description**

When performing asynchronous local evaluations, the `local_evaluation_scheduling` keyword controls how new evaluation jobs are dispatched when one completes.

The two options are:

- **dynamic**
- **static**

If the `local_evaluation_scheduling` is specified as dynamic (the default), each completed evaluation will be replaced by the next in the local evaluation queue.

If `local_evaluation_scheduling` is specified as static, each completed evaluation will be replaced by an evaluation number that is congruent modulo the `evaluation_concurrency`. This is helpful for relative node scheduling as described in Dakota/examples/parallelism. For example, assuming only asynchronous local concurrency (no MPI), if the local concurrency is 6 and job 2 completes, it will be replaced with job 8.
For the case of hybrid parallelism, static local scheduling results in evaluation replacements that are modulo the total capacity, defined as the product of the evaluation concurrency and the number of evaluation servers. Both of these cases can result in idle processors if runtimes are non-uniform, so the default dynamic scheduling is preferred when relative node scheduling is not required.

**Dynamic**

- **Keywords Area**
- **interface**
- **asynchronous**
- **local_evaluation_scheduling**
- **dynamic**

Dynamic local scheduling (sequential)

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

If the `local_evaluation_scheduling` is specified as dynamic (the default), each completed evaluation will be replaced by the next in the local evaluation queue.

**Static**

- **Keywords Area**
- **interface**
- **asynchronous**
- **local_evaluation_scheduling**
- **static**

Static local scheduling (tiled)

**Specification**

**Alias:** none

**Argument(s):** none
CHAPTER 6. KEYWORDS AREA

Description

If local_evaluation_scheduling is specified as static, each completed evaluation will be replaced by an evaluation number that is congruent modulo the evaluation_concurrency. This is helpful for relative node scheduling as described in Dakota/examples/parallelism. For example, assuming only asynchronous local concurrency (no M-P!), if the local concurrency is 6 and job 2 completes, it will be replaced with job 8.

For the case of hybrid parallelism, static local scheduling results in evaluation replacements that are modulo the total capacity, defined as the product of the evaluation concurrency and the number of evaluation servers. Both of these cases can result in idle processors if runtimes are non-uniform, so the default dynamic scheduling is preferred when relative node scheduling is not required.

analysis_concurrency

• Keywords Area
• interface
• asynchronous
• analysis_concurrency

Limit the number of analysis drivers within an evaluation that Dakota will schedule

Topics

This keyword is related to the topics:

• concurrency_and_parallelism

Specification

Alias: none

Argument(s): INTEGER

Default: local: unlimited concurrency, hybrid: no concurrency

Description

When asynchronous execution is enabled and each evaluation involves multiple analysis drivers, then the default behavior is to launch all drivers simultaneously. The analysis_concurrency keyword can be used to limit the number of concurrently run drivers.

6.5.5 evaluation_servers

• Keywords Area
• interface
• evaluation_servers

Specify the number of evaluation servers when Dakota is run in parallel
6.5. INTERFACE

Topics
This keyword is related to the topics:

- concurrency_and_parallelism

Specification
Alias: none

Argument(s): INTEGER
Default: no override of auto configure

Description
The optional evaluation_servers specification supports user override of the automatic parallel configuration for the number of evaluation servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the evaluation parallelism level. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

6.5.6 evaluation_scheduling

- Keywords Area
- interface
- evaluation_scheduling

Specify the scheduling of concurrent evaluations when Dakota is run in parallel

Specification
Alias: none

Argument(s): none
Default: automatic (see discussion)

<table>
<thead>
<tr>
<th>Required/-Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>Specify a dedicated master partition for parallel evaluation scheduling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>peer</td>
<td>Specify a peer partition for parallel evaluation scheduling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description
When Dakota is run in parallel, the partition type and scheduling for the evaluation servers are determined automatically. If these settings are undesirable, they may be overridden by the user using the evaluation_scheduling keyword.

The partition type and scheduling are
master

- Keywords Area
- interface
- evaluation_scheduling
- master

Specify a dedicated master partition for parallel evaluation scheduling

**Topics**

This keyword is related to the topics:

- concurrency_and_parallelism

**Specification**

Alias: none

**Argument(s):** none

**Description**

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the evaluation servers. This reduces the number of processors available to create servers by 1.

peer

- Keywords Area
- interface
- evaluation_scheduling
- peer

Specify a peer partition for parallel evaluation scheduling

**Topics**

This keyword is related to the topics:

- concurrency_and_parallelism

**Specification**

Alias: none

**Argument(s):** none
### Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to evaluation servers. The scheduling, static or dynamic, must also be specified.

**dynamic**

- **Keywords Area**
- interface
- evaluation_scheduling
- peer
- dynamic

Specify dynamic scheduling in a peer partition when Dakota is run in parallel.

### Topics

This keyword is related to the topics:

- concurrency_and_parallelism

### Specification

_Alias:_ none

_Argument(s):_ none

_Default:_ dynamic (see discussion)

### Description

In _dynamic_ scheduling, evaluations are assigned to servers as earlier evaluations complete. Dynamic scheduling is advantageous when evaluations are of uneven duration.
static

- Keywords Area
- interface
- evaluation_scheduling
- peer
- static

Specify static scheduling in a peer partition when Dakota is run in parallel.

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none

Argument(s): none

Description

In static scheduling, all available evaluations are assigned to servers in a predetermined fashion. Each completed evaluation is replaced with one congruent modulo the evaluation concurrency. For example, with 6 servers, eval number 2 will be replaced by eval number 8.

6.5.7 processors_per_evaluation

- Keywords Area
- interface
- processors_per_evaluation

Specify the number of processors per evaluation server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none

Argument(s): INTEGER
6.5. INTERFACE

Description
The optional processors_per_evaluation specification supports user override of the automatic parallel configuration for the number of processors in each evaluation server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the evaluation parallelism level. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

6.5.8 analysis_servers

- Keywords Area
- interface
- analysis_servers

Specify the number of analysis servers when Dakota is run in parallel

Topics
This keyword is related to the topics:
- concurrency_and_parallelism

Specification

Alias: none

Argument(s): INTEGER
Default: no override of auto configure

Description
The optional analysis_servers specification supports user override of the automatic parallel configuration for the number of analysis servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the analysis parallelism level. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [4] for additional information.

6.5.9 analysis_scheduling

- Keywords Area
- interface
- analysis_scheduling

Specify the scheduling of concurrent analyses when Dakota is run in parallel

Topics
This keyword is related to the topics:
- concurrency_and_parallelism
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: no override of auto configure

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>master</td>
<td>Specify a dedicated master partition for parallel analysis scheduling</td>
</tr>
</tbody>
</table>

peers

Specify a peer partition for parallel analysis scheduling

Description

When Dakota is run in parallel, the partition type for the analysis servers is determined automatically. If this setting is undesirable, it may be overridden by the user using the analysis.scheduling keyword.

The partition type and scheduling are

master

- Keywords Area
- interface
- analysis.scheduling
- master

Specify a dedicated master partition for parallel analysis scheduling

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition.

In a dedicated master partition, one processor (the "master") dynamically schedules work on the analysis servers. This reduces the number of processors available to create servers by 1.
6.6. RESPONSES

peer

- Keywords Area
- interface
- analysis_scheduling
- peer

Specify a peer partition for parallel analysis scheduling

Topics

This keyword is related to the topics:

- concurrency_and_parallelism

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to analysis servers. Note that unlike the case of evaluation_scheduling, it is not possible to specify static or dynamic.

6.6 responses

- Keywords Area
- responses

Description of the model output data returned to Dakota upon evaluation of an interface.

Topics

This keyword is related to the topics:

- block

Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Optional</th>
<th>id_responses</th>
<th>Name the response block, helpful when there are multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>descriptors</td>
<td>Labels for the responses</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>objective_functions</td>
<td>Response type suitable for optimization</td>
</tr>
<tr>
<td></td>
<td>calibration_terms</td>
<td>Response type suitable for calibration or least squares</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>response_functions</td>
<td>Generic response type</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>no_gradients</td>
<td>Gradients will not be used</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>analytic_gradients</td>
<td>Analysis driver will return gradients</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>mixed_gradients</td>
<td>Gradients are needed and will be obtained from a mix of numerical and analytic sources</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>numerical_gradients</td>
<td>Gradients are needed and will be approximated by finite differences</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>no_hessians</td>
<td>Hessians will not be used</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>numerical_hessians</td>
<td>Hessians are needed and will be approximated by finite differences</td>
</tr>
<tr>
<td>Required (Choose One)</td>
<td>quasi_hessians</td>
<td>Hessians are needed and will be approximated by secant updates (BFGS or SR1) from a series of gradient evaluations</td>
</tr>
</tbody>
</table>
6.6. RESPONSES

| analytic_hessians | Hessians are needed and are available directly from the analysis driver |
| mixed_hessians    | Hessians are needed and will be obtained from a mix of numerical, analytic, and “quasi” sources |

Description

The responses specification in a Dakota input file indicates the types of data that can be returned by an interface when invoked during Dakota’s execution. The specification includes three groups and two optional keywords.

**Group 1** is related to the type and number of responses expected by Dakota.

The specification must be one of three types:

1. objective and constraint functions
2. calibration (least squares) terms and constraint functions
3. a generic response functions specification.

These correspond to optimization, least squares, and uncertainty quantification methods, respectively. The response type chosen from Group 1 should be consistent with the iterative technique called for in the method specification. Certain general-purpose iterative techniques, such as parameter studies and design of experiments methods, can be used with any of these data sets.

Each type of response has additional required and optional keywords.

**Group 2** is related to the availability of first derivatives (gradient vectors) for the response functions.

The gradient specification also links back to the iterative method used. Gradients commonly are needed when the iterative study involves gradient-based optimization, local reliability analysis for uncertainty quantification, or local sensitivity analysis. They can optionally be used to build some types of surrogate models.

**Group 3** is related to the availability of second derivatives (Hessian matrices) for the response functions.

Hessian availability for the response functions is similar to the gradient availability specifications, with the addition of support for ”quasi-Hessians”. The Hessian specification also links back to the iterative method in use; Hessians commonly would be used in gradient-based optimization by full Newton methods or in reliability analysis with second-order limit state approximations or second-order probability integrations.

Examples

Several examples follow. The first example shows an optimization data set containing an objective function and two nonlinear inequality constraints. These three functions have analytic gradient availability and no Hessian availability.

```
responses
  objective_functions = 1
  nonlinear_inequality_constraints = 2
  analytic_gradients
  no_hessians
```
The next example shows a typical specification for a calibration data set. The six residual functions will have numerical gradients computed using the dakota finite differencing routine with central differences of 0.1% (plus/minus delta relative to current variables value = .001*value).

```plaintext
responses
calibration_terms = 6
numerical_gradients
    method_source dakota
    interval_type central
    fd_gradient_step_size = .001
    no_hessians
```

The last example shows a generic specification that could be used with a nondeterministic sampling iterator. The three response functions have no gradient or Hessian availability; therefore, only function values will be used by the iterator.

```plaintext
responses
    response_functions = 3
    no_gradients
    no_hessians
```

Parameter study and design of experiments iterators are not restricted in terms of the response data sets which may be catalogued; they may be used with any of the function specification examples shown above.

**Theory**

Responses specify the total data set that is available for use by the method over the course of iteration. This is distinguished from the data subset described by an active set vector (see Dakota File Data Formats in the Users Manual [Adams et al., 2010]) indicating the particular subset of the response data needed for a particular function evaluation. Thus, the responses specification is a broad description of the data to be used during a study whereas the active set vector indicates the subset currently needed.

### 6.6.1 id_responses

- **Keywords Area**
- **responses**
- **id_responses**

Name the response block, helpful when there are multiple

**Topics**

This keyword is related to the topics:

- **block_identifier**

**Specification**

- **Alias:** none
  - **Argument(s):** STRING
  - **Default:** use of last responses parsed
6.6. **RESPONSES**

### Description

The optional block identifier `id.responses` specifies a string to uniquely identify a particular responses specification (typically used when there are multiple present). A model can then specify or point to this response set by specifying the same string in its `responses_pointer` specification. For example, a model whose specification contains `responses_pointer = 'R1'` will use a responses set with `id.responses = 'R1'`.

If the `id.responses` specification is omitted, a particular responses specification will be used by a model only if that model omits specifying a `responses_pointer` and if the responses set was the last set parsed (or is the only set parsed). In common practice, if only one responses set exists, then `id.responses` can be safely omitted from the responses specification and `responses_pointer` can be omitted from the model specification(s), since there is no potential for ambiguity in this case.

#### 6.6.2 descriptors

- **Keywords Area**
- **responses**
- **descriptors**

Labels for the responses

### Specification

**Alias:** `response_descriptors`

- **Argument(s):** STRINGLIST
- **Default:** root strings plus numeric identifiers

### Description

The optional response labels specification `descriptors` is a list of strings which will be printed in Dakota output to identify the values for particular response functions.

Note that the ordering of responses and descriptors in the input currently must match the order of the values returned to Dakota in a `results_file`. See the example below.

The default descriptor strings use a root string plus a numeric identifier. This root string is

- "obj_fn" for objective functions
- "least_sq_term" for least squares terms
- "response_fn" for generic response functions
- "nln_ineq_con" for nonlinear inequality constraints
- "nln_eq_con" for nonlinear equality constraints

### Examples

Note that the descriptors currently must match the order of the values in the results file; they are not used to validate the returned data. For example, if the responses block contains:

```
  \c descriptors 'x1' 'x2' 'x3'
```

and the results file contains
Then Dakota will understand the returned data to be:

\[
\begin{align*}
\text{x1} &= 4 \\
\text{x2} &= 5 \\
\text{x3} &= 6
\end{align*}
\]

### 6.6.3 objective\_functions

- Keywords Area
- responses
- objective\_functions

Response type suitable for optimization

## Specification

**Alias:** num\_objective\_functions

**Argument(s):** INTEGER

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>sense</td>
<td>sense</td>
<td>Whether to minimize or maximize each objective function</td>
</tr>
<tr>
<td>Optional</td>
<td>primary_scale_types</td>
<td>primary_scale_types</td>
<td>Choose a scaling type for each response</td>
</tr>
<tr>
<td>Optional</td>
<td>primary_scales</td>
<td>primary_scales</td>
<td>Supply a characteristic value to scale each response</td>
</tr>
<tr>
<td>Optional</td>
<td>weights</td>
<td>weights</td>
<td>Specify weights for each objective function</td>
</tr>
<tr>
<td>Optional</td>
<td>nonlinear_inequality_constraints</td>
<td>nonlinear_inequality_constraints</td>
<td>Group to specify nonlinear inequality constraints</td>
</tr>
</tbody>
</table>
### Description

The `objective_functions` keyword specifies the number of objective functions returned to Dakota. The number of objective functions must be 1 or greater.

#### Constraints

Constraints

The keywords `nonlinear_inequality_constraints` and `nonlinear_equality_constraints` specify the number of nonlinear inequality constraints, and nonlinear equality constraints, respectively. When interfacing to external applications, the responses must be returned to Dakota in this order: objective functions, nonlinear_inequality_constraints, then nonlinear_equality_constraints. Any linear constraints present in an application need only be input to an optimizer at start up and do not need to be part of the data returned on every function evaluation. These are therefore specified in the method block. Bounds on the design variables are specified in the variables block.

#### Optional Keywords

The optional keywords relate to scaling the objective functions (for better numerical results), formulating the problem as minimization or maximization, and dealing with multiple objective functions. If scaling is used, it is applied before multi-objective weighted sums are formed.

#### See Also

These keywords may also be of interest:

- `calibration_terms`
- `response_functions`
- `method`
- `variables`

#### sense

- Keywords Area
- responses
- objective_functions
- sense

Whether to minimize or maximize each objective function

#### Specification

**Alias:** none

**Argument(s):** STRINGLIST

**Default:** vector values = 'minimize'
### Description

The `sense` keyword is used to declare whether each objective function should be minimized or maximized. The argument options are:

- "minimization"
- "maximization" These can be abbreviated to "min" and "max".

The number of strings should either be equal to the number of objective functions, or one. If a single string is specified it will apply to each objective function.

#### primary_scale_types

- **Keywords Area**
- **responses**
- **objective_functions**
- **primary_scale_types**

Choose a scaling type for each response

#### Specification

**Alias:** objective_function_scale_types

**Argument(s):** STRINGLIST

**Default:** no scaling

### Description

The `primary_scale_types` keyword specifies one of number of primary functions strings indicating the scaling type for each response value in methods that support scaling, when scaling is enabled.

See the `scaling` keyword in the method section for details on how to use this keyword. Note that primary response functions (objective, calibration, or response functions) cannot be automatically scaled due to lack of bounds, so valid scale types are 'none', 'value' and 'log'.

#### primary_scales

- **Keywords Area**
- **responses**
- **objective_functions**
- **primary_scales**

Supply a characteristic value to scale each response

#### Specification

**Alias:** objective_function_scales

**Argument(s):** REALLIST

**Default:** 1.0 (no scaling)
Description

Each entry in `primary_scales` is a user-specified nonzero characteristic value to scale each response. The argument may be of length 1 or the number of primary response functions. See the `scaling` keyword in the `method` section for details on how to use this keyword.

**weights**

- Keywords Area
- responses
- objective_functions
- weights

Specify weights for each objective function.

Specification

**Alias:** multi_objective_weights

**Argument(s):** REALLIST

**Default:** equal weights

**Description**

If the number of objective functions is greater than 1, then a `weights` specification provides a simple weighted-sum approach to combining multiple objectives into a single objective:

\[ f = \sum_{i=1}^{n} w_i f_i \]

If weights are not specified, then each response is given equal weighting:

\[ f = \frac{\sum_{i=1}^{n} f_i}{n} \]

where, in both of these cases, a "minimization" sense will retain a positive weighting for a minimizer and a "maximization" sense will apply a negative weighting.

**nonlinear_inequality_constraints**

- Keywords Area
- responses
- objective_functions
- nonlinear_inequality_constraints

Group to specify nonlinear inequality constraints.

Specification

**Alias:** num_nonlinear_inequality_constraints

**Argument(s):** INTEGER

**Default:** 0
CHAPTER 6. KEYWORDS AREA

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>lower_bounds</td>
<td>Specify minimum values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>upper_bounds</td>
<td>Specify maximum values</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>scale_types</td>
<td>Choose how each constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>scales</td>
<td>Characteristic values for scaling</td>
</tr>
</tbody>
</table>

Description

The `lower_bounds` and `upper_bounds` specifications provide the lower and upper bounds for 2-sided nonlinear inequalities of the form

\[ g_l \leq g(x) \leq g_u \]

The defaults for the inequality constraint bounds are selected so that one-sided inequalities of the form

\[ g(x) \leq 0.0 \]

result when there are no user constraint bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than `+bigRealBoundSize` (1.e+30, as defined in Minimizer) are treated as `+infinity` and any lower bound values less than `-bigRealBoundSize` are treated as `-infinity`. This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since `-DBL_MAX < -bigRealBoundSize`). The same approach is used for nonexistent linear inequality bounds and for nonexistent design variable bounds.

The `scale_types` and `scales` keywords are related to scaling of \( g(x) \). See the scaling keyword in the method section for details.

**lower_bounds**

- **Keywords Area**
- **responses**
- **objective_functions**
- **nonlinear_inequality_constraints**
- **lower_bounds**

Specify minimum values

**Specification**

Alias: nonlinear_inequality_lower_bounds

Argument(s): REALLIST

Default: vector values = -infinity

**Description**

Specify minimum values
6.6. RESPONSES

upper_bounds

- Keywords Area
- responses
- objective_functions
- nonlinear_inequality_constraints
- upper_bounds

Specify maximum values

Specification

Alias: nonlinear_inequality_upper_bounds
Argument(s): REALLIST
Default: vector values = 0.

Description

Specify maximum values

scale_types

- Keywords Area
- responses
- objective_functions
- nonlinear_inequality_constraints
- scale_types

Choose how each constraint is scaled

Specification

Alias: nonlinear_inequality_scale_types
Argument(s): STRINGLIST
Default: no scaling

Description

See the scaling keyword in the method section for details on how to use this keyword.
scales

- Keywords Area
- responses
- objective_functions
- nonlinear_inequality_constraints
- scales

Characteristic values for scaling

**Specification**

**Alias:** nonlinear_inequality_scales

**Argument(s):** REALLIST

**Default:** 1.0 (no scaling)

**Description**

See the scaling keyword in the method section for details on how to use this keyword.

**nonlinear_inequality_constraints**

- Keywords Area
- responses
- objective_functions
- nonlinear_inequality_constraints

Group to specify nonlinear equality constraints

**Specification**

**Alias:** num_nonlinear_inequality_constraints

**Argument(s):** INTEGER

**Default:** 0

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>targets</td>
<td>Target values for the nonlinear equality constraint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scale_types</td>
<td>Choose how each constraint is scaled</td>
</tr>
</tbody>
</table>
Optional scales | Characteristic values for scaling

**Description**

The `targets` specification provides the targets for nonlinear equalities of the form

\[ g(x) = g_t \]

and the defaults for the equality targets enforce a value of 0.0 for each constraint

\[ g(x) = 0.0 \]

The `scale_types` and `scales` keywords are related to scaling of \( g(x) \). See the scaling keyword in the `method` section for details.

`targets`
- Keywords Area
- responses
- objective_functions
- nonlinear_equality_constraints
- targets
  Target values for the nonlinear equality constraint

**Specification**

**Alias:** nonlinear_equality_targets

**Argument(s):** REALIST

**Default:** vector values = 0.0.

**Description**

The `targets` specification provides the targets for nonlinear equalities of the form

\[ g(x) = g_t \]

and the defaults for the equality targets enforce a value of 0.0 for each constraint:

\[ g(x) = 0.0 \]

`scale_types`
- Keywords Area
- responses
- objective_functions
- nonlinear_equality_constraints
- `scale_types`
  Choose how each constraint is scaled
CHAPTER 6. KEYWORDS AREA

Specification

Alias: nonlinear_equality_scale_types

Argument(s): STRINGLIST

Default: no scaling

Description

See the scaling keyword in the method section for details on how to use this keyword.

scales

- Keywords Area
- responses
- objective_functions
- nonlinear_equality_constraints
- scales

Characteristic values for scaling

Specification

Alias: nonlinear_equality_scales

Argument(s): REALLIST

Default: 1.0 (no scaling)

Description

See the scaling keyword in the method section for details on how to use this keyword.

scalar_objectives

- Keywords Area
- responses
- objective_functions
- scalar_objectives

Number of scalar objective functions

Specification

Alias: num_scalar_objectives

Argument(s): INTEGER
6.6. RESPONSES

Description

This keyword describes the number of scalar objective functions. It is meant to be used in conjunction with field_objectives, which describes the number of field objectives functions. The total number of objective functions, both scalar and field, is given by objective_functions. If only scalar objective functions are specified, it is not necessary to specify the number of scalar terms explicitly: one can simply say objective_functions = 5 and get 5 scalar objectives. However, if there are three scalar objectives and 2 field objectives, then objective_functions = 5 but scalar_objectives = 3 and field_objectives = 2.

Objective functions are responses that are used with optimization methods in Dakota. Currently, each term in a field objective is added to the total objective function presented to the optimizer. For example, if you have one field objective with 100 terms (e.g. a time-temperature trace with 100 time points and 100 corresponding temperature points), the 100 temperature values will be added to create the overall objective.

See Also

These keywords may also be of interest:

- field_objectives

field_objectives

- Keywords Area
- responses
- objective_functions
- field_objectives

Number of field objective functions

Specification

Alias: num_field_objectives

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional   | Group         |               | Description      |
| Required   | lengths       | num_coordinates_ | Lengths of field |
|            |              | per_field      | responses        |
| Optional   | num_coordinates_ |              | Number of       |
|            |              | read_field_     | independent      |
|            |              | coordinates    | coordinates for  |
|            |              |                 | field responses  |
|            |              |                 | Add context to   |
|            |              |                 | data: flag to    |
|            |              |                 | indicate that    |
|            |              |                 | field coordinates |
|            |              |                 | should be read   |


CHAPTER 6. KEYWORDS AREA

Description

This keyword describes the number of field objective functions. A field function is a set of related response values collected over a range of independent coordinate values which may or may not be specified by the user. For example, voltage over time would be a field function, where voltage is the field_objective and time is the independent coordinate. Similarly, temperature over time and space would be a field response, where the independent coordinates would be both time and spatial coordinates such as (x,y) or (x,y,z), depending on the application. The main difference between scalar objectives and field objectives is that for field data, we plan to implement methods that take advantage of the correlation or relationship between the field values.

Note that if there is one field_objective, and it has length 100 (meaning 100 values), then the user’s simulation code must return 100 values. Also, if there are both scalar and field objectives, the user should specify the number of scalar objectives as scalar_objectives. If there are only field objectives, it still is necessary to specify both objective_functions = NN and field_objectives = NN, where NN is the number of field objectives.

Objective functions are responses that are used with optimization methods in Dakota. Currently, each term in a field objective is added to the total objective function presented to the optimizer. For example, if you have one field objective with 100 terms (e.g. a time-temperature trace with 100 time points and 100 corresponding temperature points), the 100 temperature values will be added to create the overall objective.

See Also

These keywords may also be of interest:

- scalar_objectives

lengths

- Keywords Area
- responses
- objective_functions
- field_objectives
- lengths

Lengths of field responses

Specification

Alias: none

Argument(s): INTEGERLIST

Description

This keyword describes the lengths of each field response. It is an integer vector of length field_responses. For example, if the field_responses = 2, an example would be lengths = 50 200, indicating that the first field response has 50 field elements but the second one has 200. The coordinate values (e.g. the independent variables) corresponding to these field responses are read in files labeled response_descriptor.coords.
6.6. RESPONSES

See Also
These keywords may also be of interest:

• field_responses

num_coordinates_per_field

• Keywords Area
• responses
• objective_functions
• field_objectives
• num_coordinates_per_field

Number of independent coordinates for field responses

Specification
Alias: none
Argument(s): INTEGERLIST

Description
This keyword describes the number of independent coordinates for each field response. It is an integer vector of length field_responses. For example, if the field_responses = 2, an example would be num_coordinates_per_field = 2 1 means that the first field response has two sets of independent coordinates (perhaps x, y locations), but the second response only has one (for example, time where the field response is only dependent upon time). The actual coordinate values (e.g. the independent variables) corresponding to these field responses are defined in a file call response_descriptor.coords, where response_descriptor is the name of the individual field.

See Also
These keywords may also be of interest:

• field_responses

read_field_coordinates

• Keywords Area
• responses
• objective_functions
• field_objectives
• read_field_coordinates

Add context to data: flag to indicate that field coordinates should be read
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): none

Description

Field coordinates specify independent variables (e.g. spatial or temporal coordinates) upon which the field depends. For example, the voltage level above might be a function of time, so time is the field coordinate. If the user has field coordinates to read, they need to specify `read_field_coordinates`. The field coordinates will then be read from a file named `response_descriptor.coords`, where `response_descriptor` is the user-provided descriptor for the field response. The number of columns in the coords file should be equal to the number of field coordinates.

6.6.4 calibration_terms

- Keywords Area
- responses
- calibration_terms

Response type suitable for calibration or least squares

Specification

Alias: least_squares_terms num_least_squares_terms
  Argument(s): INTEGER

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>scalar_calibration_terms</td>
<td>Number of scalar calibration terms</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>field_calibration_terms</td>
<td>Number of field calibration terms</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>primary_scale_types</td>
<td>Choose a scaling type for each response</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>primary_scales</td>
<td>Supply a characteristic value to scale each response</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>weights</td>
<td>Apply different weights to each response</td>
</tr>
</tbody>
</table>
6.6. RESPONSES

<table>
<thead>
<tr>
<th>Optional (Choose One)</th>
<th>Group 1</th>
<th>calibration_data</th>
<th>Supply calibration data in the case of field data or mixed data (both scalar and field data).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>calibration_data-file</td>
<td>Specify a text file containing calibration data for scalar responses.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>nonlinear_inequality_constraints</td>
<td>Group to specify nonlinear inequality constraints.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>nonlinear_equality_constraints</td>
<td>Group to specify nonlinear equality constraints.</td>
</tr>
</tbody>
</table>

**Description**

Responses for a calibration study are specified using `calibration_terms` and optional keywords for weighting/scaling, data, and constraints. In general when calibrating, Dakota automatically tunes parameters \( \theta \) to minimize discrepancies or residuals between the model and the data:

\[
R_i = y_i^{\text{Model}}(\theta) - y_i^{\text{Data}}.
\]

There are two use cases:

- If `calibration_data_file` is NOT specified, then each of the calibration terms returned to Dakota through the interface is a residual \( R_i \) to be driven toward zero.

- If `calibration_data_file` IS specified, then each of the calibration terms returned to Dakota must be a response \( y_i^{\text{Model}}(\theta) \), which Dakota will difference with the data in the specified data file.

**Constraints**

The keywords `nonlinear_inequality_constraints`, and `nonlinear_equality_constraints` specify the number of nonlinear inequality constraints, and nonlinear equality constraints, respectively. When interfacing to external applications, the responses must be returned to Dakota in this order: calibration terms, nonlinear_inequality_constraints, then nonlinear_equality_constraints.

Any linear constraints present in an application need only be input to an optimizer at start up and do not need to be part of the data returned on every function evaluation. These are therefore specified in the method block.

**Optional Keywords**

The optional keywords relate to scaling responses (for better numerical results), dealing with multiple residuals, and importing data.

See the `scaling` keyword in the method section for more details on scaling. If scaling is specified, then it is applied to each residual prior to squaring:

\[
f = \sum_{i=1}^{n} w_i \left( \frac{y_i^{\text{Model}} - y_i^{\text{Data}}}{s_i} \right)^2.
\]
In the case where experimental data uncertainties are supplied, then the weights are automatically defined to be the inverse of the experimental variance:

\[
f = \frac{1}{\sigma_i^2} \left( \frac{y_i^{Model} - y_i^{Data}}{s_i} \right)^2
\]

**Theory**

Dakota calibration terms are typically used to solve problems of parameter estimation, system identification, and model calibration/inversion. Local least squares calibration problems are most efficiently solved using special-purpose least squares solvers such as Gauss-Newton or Levenberg-Marquardt; however, they may also be solved using any general-purpose optimization algorithm in Dakota. While Dakota can solve these problems with either least squares or optimization algorithms, the response data sets to be returned from the simulator are different when using objective functions versus calibration terms.

Least squares calibration involves a set of residual functions, whereas optimization involves a single objective function (sum of the squares of the residuals), i.e.,

\[
f = \sum_{i=1}^{n} R_i^2 = \sum_{i=1}^{n} (y_i^{Model}(\theta) - y_i^{Data})^2
\]

where \( f \) is the objective function and the set of \( R_i \) are the residual functions, most commonly defined as the difference between a model response and data. Therefore, function values and derivative data in the least squares case involve the values and derivatives of the residual functions, whereas the optimization case involves values and derivatives of the sum of squares objective function. This means that in the least squares calibration case, the user must return each of \( n \) residuals separately as a separate calibration term. Switching between the two approaches sometimes requires different simulation interfaces capable of returning the different granularity of response data required, although Dakota supports automatic recasting of residuals into a sum of squares for presentation to an optimization method. Typically, the user must compute the difference between the model results and the observations when computing the residuals. However, the user has the option of specifying the observational data (e.g. from physical experiments or other sources) in a file.

**See Also**

These keywords may also be of interest:

- objective_functions
- response_functions

**scalar_calibration_terms**

- Keywords Area
- responses
- calibration_terms
- scalar_calibration_terms

Number of scalar calibration terms
6.6. RESPONSES

Specification

Alias: none
Argument(s): INTEGER

Description

This keyword describes the number of scalar calibration terms. It is meant to be used in conjunction with field_calibra-
tion_terms, which describes the number of field calibration terms. The total number of calibration terms, both scalar and field, is given by calibration_terms. If only scalar calibration terms are specified, it is not necessary to specify the number of scalar terms explicitly: one can simply say calibration_terms = 5 and get 5 scalar terms. However, if there are three scalar terms and 2 field terms, then calibration_terms = 5 but scalar_calibration_terms = 3 and field_calibration_terms = 2.

Calibration terms are responses that are used with calibration methods in Dakota, such as least squares optimizers. Currently, each scalar term is added to the total sum-of-squares error function presented to the optimizer. However, each individual field value is added as well. For example, if you have one field calibration term with length 100 (e.g. a time-temperature trace with 100 time points and 100 temperature points), the 100 temperature values will be added to create the overall sum-of-squares error function used in calibration.

See Also

These keywords may also be of interest:

- field_calibration_terms

field_calibration_terms

- Keywords Area
- responses
- calibration_terms
- field_calibration_terms

Number of field calibration terms

Specification

Alias: none
Argument(s): INTEGER

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>lengths</td>
<td>num_coordinates_-per_field</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Lengths of field responses
- Number of independent coordinates for field responses
CHAPTER 6. KEYWORDS AREA

Optional read_field_coordinates
Add context to data: flag to indicate that field coordinates should be read

Description

This keyword describes the number of field calibration terms. A set of field calibration terms is a set of related response values collected over a range of independent coordinate values which may or may not be specified by the user. For example, voltage over time would be a field function, where voltage is the field_objective and time is the independent coordinate. Similarly, temperature over time and space would be a field response, where the independent coordinates would be both time and spatial coordinates such as (x,y) or (x,y,z), depending on the application. The main difference between scalar calibration terms and field calibration terms is that for field data, we plan to implement methods that take advantage of the correlation or relationship between the field values. For example, with calibration, if we want to calibrate parameters that result in a good model fit to a time-temperature curve, we may have to do some interpolation between the experimental data and the simulation data. That capability requires knowledge of the independent coordinates.

Note that if there is one field_calibration_terms, and it has length 100 (meaning 100 values), then the user’s simulation code must return 100 values. Also, if there are both scalar and field calibration, the user should specify the number of scalar terms as scalar_calibration_terms. If there are only field calibration terms, it still is necessary to specify both field_calibration_terms = NN and calibration_terms = NN, where NN is the number of field calibration terms.

Calibration terms are responses that are used with calibration methods in Dakota, such as least squares optimizers. Currently, each scalar term is added to the total sum-of-squares error function presented to the optimizer. However, each individual field value is added as well. For example, if you have one field calibration term with length 100 (e.g. a time-temperature trace with 100 time points and 100 temperature points), the 100 temperature values will be added to create the overall sum-of-squares error function used in calibration. We have an initial capability to interpolate the field data from the user’s simulation to the experimental data. For example, if the user has thermocouple readings at 20 time points, it will be an experimental field response with 20 time points and 20 temperature values. Dakota takes the 100 simulation time-temperature values (from the example above) and interpolates those to the 20 experimental points, to create 20 residual terms (simulation minus experimental data points) that will be used in calibration.

See Also

These keywords may also be of interest:

- scalar_calibration_terms
- lengths

Keywords Area
- responses
- calibration_terms
- field_calibration_terms
- lengths

Lengths of field responses
6.6. RESPONSES

Specification

Alias: none
Argument(s): INTEGERLIST

Description

This keyword describes the lengths of each field response. It is an integer vector of length field_responses. For example, if the field_responses = 2, an example would be lengths = 50 200, indicating that the first field response has 50 field elements but the second one has 200. The coordinate values (e.g. the independent variables) corresponding to these field responses are read in files labeled response_descriptor.coords.

See Also

These keywords may also be of interest:

- field_responses

num_coordinates_per_field

- Keywords Area
- responses
- calibration_terms
- field_calibration_terms
- num_coordinates_per_field

Number of independent coordinates for field responses

Specification

Alias: none
Argument(s): INTEGERLIST

Description

This keyword describes the number of independent coordinates for each field response. It is an integer vector of length field_responses. For example, if the field_responses = 2, an example would be num_coordinates_per_field = 2 1 means that the first field response has two sets of independent coordinates (perhaps x, y locations), but the second response only has one (for example, time where the field response is only dependent upon time). The actual coordinate values (e.g. the independent variables) corresponding to these field responses are defined in a file call response_descriptor.coords, where response_descriptor is the name of the individual field.

See Also

These keywords may also be of interest:

- field_responses
read_field_coordinates

- Keywords Area
- responses
- calibration_terms
- field_calibration_terms
- read_field_coordinates

Add context to data: flag to indicate that field coordinates should be read

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Field coordinates specify independent variables (e.g. spatial or temporal coordinates) upon which the field depends. For example, the voltage level above might be a function of time, so time is the field coordinate. If the user has field coordinates to read, they need to specify `read_field_coordinates`. The field coordinates will then be read from a file named `response_descriptor.coords`, where `response_descriptor` is the user-provided descriptor for the field response. The number of columns in the coords file should be equal to the number of field coordinates.

primary_scale_types

- Keywords Area
- responses
- calibration_terms
- primary_scale_types

Choose a scaling type for each response

**Specification**

**Alias:** calibration_term_scale_types least_squares_term_scale_types

**Argument(s):** STRINGLIST

**Default:** no scaling

**Description**

The `primary_scale_types` keyword specifies one of number of primary functions strings indicating the scaling type for each response value in methods that support scaling, when scaling is enabled.

See the `scaling` keyword in the `method` section for details on how to use this keyword. Note that primary response functions (objective, calibration, or response functions) cannot be automatically scaled due to lack of bounds, so valid scale types are 'none', 'value' and 'log'.
6.6. RESPONSES

**primary_scales**

- Keywords Area
- responses
- calibration_terms
- primary_scales

Supply a characteristic value to scale each response

**Specification**

**Alias:** calibration_term_scales least_squares_term_scales

**Argument(s):** REALLIST

**Default:** 1.0 (no scaling)

**Description**

Each entry in primary_scales is a user-specified nonzero characteristic value to scale each response.

The argument may be of length 1 or the number of primary response functions. See the scaling keyword in the method section for details on how to use this keyword.

**weights**

- Keywords Area
- responses
- calibration_terms
- weights

Apply different weights to each response

**Specification**

**Alias:** calibration_weights least_squares_weights

**Argument(s):** REALLIST

**Default:** equal weights

**Description**

The weights specification provides a means to specify a relative emphasis among the vector of squared residuals through multiplication of these squared residuals by a vector of weights:

\[ f = \sum_{i=1}^{n} w_i R_i^2 = \sum_{i=1}^{n} w_i (y_i^M - y_i^O)^2 \]
calibration_data

- Keywords Area
- responses
- calibration_terms
- calibration_data

Supply calibration data in the case of field data or mixed data (both scalar and field data).

### Specification

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/-</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>num_experiments</td>
<td></td>
<td>Add context to data: number of different experiments</td>
</tr>
<tr>
<td>Optional</td>
<td>num_config_variables</td>
<td></td>
<td>Add context to data: number of configuration variables.</td>
</tr>
<tr>
<td>Optional</td>
<td>variance_type</td>
<td></td>
<td>Add context to experiment data description by specifying the type of experimental error.</td>
</tr>
<tr>
<td>Optional</td>
<td>scalar_data_file</td>
<td></td>
<td>Specify a scalar data file to complement field data files (mixed case)</td>
</tr>
<tr>
<td>Optional</td>
<td>interpolate</td>
<td></td>
<td>Flag to indicate interpolation of simulation values.</td>
</tr>
</tbody>
</table>

### Description

calibration_data specifies a keyword block that indicates that Dakota should read in various types of experimental data. This block is primarily to support the reading of field calibration data.

The user will specify the number of experiments, num_experiments. If this is not specified, it is assumed there is only one experiment.

For each experiment, there are four main types of data related to fields:
6.6. RESPONSES

- values.
  These are the values of the experiment field (e.g., temperature values, voltage levels, etc.) These MUS-T be specified in a file named response_descriptor.NUM.dat, where NUM is replaced by the number of the experiment (1, 2, ...) and response_descriptor is replaced by the actual response_descriptor the user has specified. For example, if response_descriptor = voltage, then the voltage field values for experiment 1 should be in a file labeled voltage.1.dat. The field values should be in a column (e.g. one field value per row).

- field coordinates.
  Field coordinates specify independent variables (e.g. spatial or temporal coordinates) upon which the field depends. For example, the voltage level above might be a function of time, so time is the field coordinate. For the calibration data, the field coordinates are read from a file named response_descriptor.NUM.coords, similar to the pattern for field values. The number of columns in the coords file should be equal to the number of field coordinates.

- variance terms.
  The user needs to specify variance_type, which defines the type of experimental measurement error. This is a string list, with one variance type specified for each scalar and each field. The available types are 'none' (no variance is specified), 'scalar' (one scalar value is specified which is applicable to all of the measurements in the field), 'diagonal' (a column vector is provided which contains the number of elements of the experimental data field. Each element of the vector contains the experimental measurement variance corresponding to that particular field measurement. For example, for the fifth term in the field, the fifth term in the diagonal vector would have the variance of that measurement), and 'matrix'. In the matrix case, a full covariance matrix is provided, which contains the pairwise correlations between each element in the field. For each field, if the variance_type is not 'none', a data file must be provided with the name response_descriptor.NUM.sigma, which contains the appropriate variance data for that field.

- configuration variables.
  Configuration variables specify the conditions corresponding to different experiments. This is used when there is more than one experiment. If num_config_variables is positive, the configuration variables for each experiment should be placed in a file named experiment.NUM.config, where the number of items in that config file are the num_config_variables. Currently, configuration variables are not being used. However, the intent is that they will be used as state variables for the simulation (for example) and not as variables that would be calibrated.

The above description is relevant for field data (with files for field values, field coordinates, field variances). If the user also has scalar experimental data, it may be entered in the same way (e.g. one file for a scalar experimental value named scalar_response_descriptor.NUM.dat, etc.) However, the user can choose to enter the scalar data in the format that we offer for pure scalar data, where we have just one file, with the number of rows of that file equal to the number of experiments. In that case, the scalar data is in the file given by the name scalar_data_file, and the format of that file is the same as discussed in calibration_data_file.

One important feature of field data is the capability to interpolate between points in the field. For example, we may have simulation data at a set of responses y at time points t: (t_{s1}, y_{s1}), (t_{s2}, y_{s2}), etc. In this example, t is the independent coordinate for the simulation, and the simulation time and response points are denoted with subscripts s1, s2, s3, etc. If the user has experimental data that is taken at different time points: (t_{e1}, y_{e1}), (t_{e2}, y_{e2}), ..., it is necessary to interpolate the simulation data to provide estimates of the simulation response at the experimental time points to construct the residual terms (model - experiment) at the experimental time points. Dakota can perform this interpolation now. The user must specify the keyword interpolate, and also provide the field coordinates as well as field values for the experiment data. If the interpolate keyword is not specified, Dakota will assume that the simulation field data and the experiment field data is taken at the same set of independent
The number of different experiments. Dakota will expand the total number of residual terms based on the number of calibration terms and the number of experiments. For example, if the number of calibration terms are five scalars, and there are three experiments, the total number of residuals in the least squares formulation will be 15. See calibration_data or calibration_data_file.

**Specification**

**Alias:** none

**Argument(s):** INTEGER

**Default:** 1

**Description**

The number of different experiments. Dakota will expand the total number of residual terms based on the number of calibration terms and the number of experiments. For example, if the number of calibration terms are five scalars, and there are three experiments, the total number of residuals in the least squares formulation will be 15. See calibration_data or calibration_data_file.
Description

This is an optional description. If there are multiple experiments, there can be different configuration variables (e.g. experimental settings, boundary conditions, etc.) per experiment. See calibration_data or calibration_data_file. Note: as of Dakota 6.2, configuration variables are not used. However, the intent is to treat them as state variables which will be passed to the simulation, and not treated as parameters which are calibrated.

variance_type

- Keywords Area
- responses
- calibration_terms
- calibration_data
- variance_type

Add context to experiment data description by specifying the type of experimental error.

Specification

Alias: none
Argument(s): STRINGLIST
Default: none

Description

There are four options for specifying the experimental error (e.g. the measurement error in the data you provide for calibration purposes): 'none', 'scalar', 'diagonal', or 'matrix.'

If the user specifies none, Dakota will calculate a variance (sigma-squared) term. This will be a constant variance term across all of the data). If the user specifies scalar, they can provide a scalar variance per calibration term. Note that for scalar calibration terms, only ‘none’ or ‘scalar’ are options for the measurement variance. However, for field calibration terms, there are two additional options. ‘diagonal’ allows the user to provide a vector of measurement variances (one for each term in the calibration field). This vector corresponds to the diagonal of the full covariance matrix of measurement errors. If the user specifies ’matrix’, they can provide a full covariance matrix (not just the diagonal terms), where each element(i,j) of the covariance matrix represents the covariance of the measurement error between the i-th and j-th field values.

scalar_data_file

- Keywords Area
- responses
- calibration_terms
- calibration_data
- scalar_data_file

Specify a scalar data file to complement field data files (mixed case)
Specification

Alias: none
Argument(s): STRING
### Description

When calibrating both scalar and field calibration terms, to associated experimental data, the scalar data may be provided in the file named by `scalar_data_file`. This file follows the same format as: `calibration_data_file`.

#### Default Behavior

If `scalar_data_file` is omitted, all calibration data, including for scalar responses, will be read from the generic field `calibration_data_format`.

- **annotated**
  - **Keywords Area**
  - **responses**
  - **calibration_terms**
  - **calibration_data**
  - **scalar_data_file**
  - **annotated**

  Selects annotated tabular file format for experiment data

#### Topics

This keyword is related to the topics:

- **file_formats**

### Specification

**Alias:** none

**Argument(s):** none

**Default:** annotated format
**Description**

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. For experiment data files, each subsequent row contains an experiment ID, followed by data for configuration variables, observations, and/or observation errors, depending on context.

**Default Behavior**

By default, Dakota imports tabular experiment data files in annotated format. The `annotated` keyword can be used to explicitly specify this.

**Usage Tips**

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

**Examples**

Import an annotated experimental data file containing a header row, leading exp_id column, and experiment data in a calibration study

```plaintext
responses
... scalar_data_file 'shock_experiment.dat'
annotated
```

Example data file with two measured quantities, three experiments:

<table>
<thead>
<tr>
<th>exp_id</th>
<th>velocity</th>
<th>stress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18.23</td>
<td>83.21</td>
</tr>
<tr>
<td>2</td>
<td>34.14</td>
<td>93.24</td>
</tr>
<tr>
<td>3</td>
<td>22.41</td>
<td>88.92</td>
</tr>
</tbody>
</table>

**custom_annotated**

- **Keywords Area**

- **responses**

- **calibration_terms**

- **calibration_data**

- **scalar_data_file**

- **custom_annotated**

  Selects custom-annotated tabular file format for experiment data

**Topics**

This keyword is related to the topics:

- **file_formats**
6.6. RESPONSES

Specification

Alias: none
  Argument(s): none
  Default: annotated format
## Description

A custom-annotated tabular file is a whitespace-separated text file containing experiment data, including configuration variables, observations, and/or observation errors, depending on context. For experiment import, custom-annotated allows user options for whether `header` row and `exp_id` column appear in the tabular file, thus bridging freeform and (fully) annotated.

### Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. The `custom_annotated` keyword, followed by options can be used to select other formats.

### Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

## Examples

Import an experimental data file containing a header row, no leading `exp_id` column, and experiment data in a calibration study:

```plaintext
responses
    scalar_data_file 'shock_experiment.dat'
    custom_annotated header

Example data file with two measured quantities, three experiments:
```

<table>
<thead>
<tr>
<th>%</th>
<th>velocity</th>
<th>stress</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.23</td>
<td></td>
<td>83.21</td>
</tr>
<tr>
<td>34.14</td>
<td></td>
<td>93.24</td>
</tr>
<tr>
<td>22.41</td>
<td></td>
<td>88.92</td>
</tr>
</tbody>
</table>

header
- Keywords Area
- responses
- calibration_terms
6.6. RESPONSES

- calibration_data
- scalar_data_file
- custom_annotated
- header

Enable header row in custom-annotated tabular file

**Specification**

Alias: none
Argument(s): none
Default: no header

Description

See description of parent custom_annotated

exp_id

- Keywords Area
- responses
- calibration_terms
- calibration_data
- scalar_data_file
- custom_annotated
- exp_id

Enable experiment ID column in custom-annotated tabular file

**Specification**

Alias: none
Argument(s): none
Default: no exp_id column

Description

See description of parent custom_annotated
freeform

- Keywords Area
- responses
- calibration_terms
- calibration_data
- scalar_data_file
- freeform

Selects free-form tabular file format for experiment data

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. For experiment data files, each row contains data for configuration variables, observations, and/or observation errors, depending on context.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. Specify freeform to instead select this format.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import a free-form experimental data file containing raw experiment data in a calibration study

responses

... scalar_data_file 'shock_experiment.dat'

freeform

Example data file with two measured quantities, three experiments:

18.23  83.21
34.14  93.24
22.41  88.92
interpolate

- Keywords Area
- responses
- calibration_terms
- calibration_data
- interpolate

Flag to indicate interpolation of simulation values.

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

If `interpolate` is specified, Dakota will interpolate between the simulation data and the experiment data to calculate the residuals for calibration methods. Specifically, the simulation data is interpolated onto the experimental data points. So, if the simulation data is a field of length 100 with one independent coordinate, and the experiment data is of length 5 with one independent coordinate, the interpolation is done between the 100 \((t,f)\) simulation points (where \(t\) is the independent coordinate and \(f\) is the simulation field value) onto the five \((t_e,f_e)\) points to obtain the residual differences between the simulation and experiment. See `calibration_data`.

**calibration_data_file**

- Keywords Area
- responses
- calibration_terms
- calibration_data_file

Specify a text file containing calibration data for scalar responses

**Specification**

**Alias:** least_squares_data_file

**Argument(s):** STRING

**Default:** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional (Choose One)</td>
<td>tabular format (Group 1)</td>
<td>annotated</td>
<td>Selects annotated tabular file format for experiment data</td>
</tr>
</tbody>
</table>
### Description

Enables text file import of experimental observations for use in calibration, for scalar responses only. Dakota will calibrate model variables to best match these data. Key options include:

- **format**: whether the data file is in annotated, custom annotated, or freeform format
- **content**: where `num_experiments`, `num_config_variables`, and `variance_type` indicate which columns appear in the data.

While some components may be omitted, the most complete version of an annotated calibration data file could include columns corresponding to:

```
exp_id | configuration xvars | y data observations | y data variances
```

Each row in the file corresponds to an experiment or replicate observation of an experiment to be compared to the model output.

### Usage Tips

- The `calibration_data_file` used when ONLY scalar calibration terms are present. If there are field calibration terms, instead use `calibration_data`. For mixed scalar and field calibration terms, one may use the `calibration_data` specification, together with its sub-specification `scalar_data_file`, which uses the format described here.

### Simple Case

In the simplest case, no data content descriptors are specified, so the data file must contain only the $y^{Data}$ observations which represent a single experimental observation. In this case, the data file should have $N_{terms}$ columns and 1 row, where $N_{terms}$ is the value of `calibration_terms`.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom_annotated</td>
<td>Selects custom-annotated tabular file format for experiment data</td>
</tr>
<tr>
<td>freeform</td>
<td>Selects free-form tabular file format for experiment data</td>
</tr>
<tr>
<td>Optional</td>
<td><code>num_experiments</code></td>
</tr>
<tr>
<td>Optional</td>
<td><code>num_config_variables</code></td>
</tr>
<tr>
<td>Optional</td>
<td><code>variance_type</code></td>
</tr>
</tbody>
</table>
For each function evaluation, Dakota will run the analysis driver, which must return \( N_{\text{terms}} \) model responses. Then the residuals are computed as:

\[
R_i = y_i^{\text{Model}} - y_i^{\text{Data}}.
\]

These residuals can be weighted using weights.

**With experimental variances**

If information is known about the measurement error and the uncertainty in the measurement, that can be specified by sending the measurement error variance to Dakota. In this case, the keyword `variance_type` is added, followed by a string of variance types of length one or of length \( N_{\text{terms}} \), where \( N_{\text{terms}} \) is the value of `calibration_terms`. The `variance_type` for each response can be 'none' or 'scalar'. NOTE: you must specify the same `variance_type` for all scalar terms. That is, they will all be 'none' or all be 'scalar.'

For each response that has a 'scalar' variance type, each row of the datafile will now have \( N_{\text{terms}} \) of \( y \) data values followed by \( N_{\text{terms}} \) columns that specify the measurement error (in units of variance, not standard deviation of the measurement error) for \( y \) variances.

Dakota will run the analysis driver, which must return \( N_{\text{terms}} \) responses. Then the residuals are computed as:

\[
R_i = \frac{y_i^{\text{Model}} - y_i^{\text{Data}}}{\sqrt{\text{VAR}_i}}
\]

for \( i = 1 \ldots N_{\text{terms}} \).

**Fully general case**

In the most general case, the content of the data file is described by the arguments of three parameters. The parameters are optional, and defaults are described below.

- **num_experiments** (\( N_{\text{exp}} \))
  - Default: \( N_{\text{exp}} = 1 \)
  - This indicates that the data represents multiple experiments, where each experiment might be conducted with different values of configuration variables. An experiment can also be thought of as a replicate, where the experiments are run at the same values of the configuration variables.

- **num_config_variables** (\( N_{\text{cfg}} \))
  - This is not yet supported, but will specify the values of experimental conditions at which data were collected.

- **variance_type** ('none' or 'scalar')
  - This indicates if the data file contains variances for measurement error of the experimental data. The default is 'none'.

If the user does not specify `variance_type`, or if the `variance_type` = 'none', only the actual observations are specified in the `calibration_data_file`. If the user specifies `variance_type` = 'scalar', then the `calibration_data_file` must contain two times `calibration_terms`. The first `calibration_terms` columns are the experimental data, and the second `calibration_terms` columns are the experimental measurement error variance. For example, if the user has three `calibration_terms`, and specifies `variance_type` = 'scalar', then the calibration data must contain six columns. The first three columns will contain the data, and the second three columns will contain the experimental error (in units of variance) for the data in the first three columns. These variances are used to weight the residuals in the sum-of-squares objective.

A more advanced use of the `calibration_data_file` might specify `num_experiments \( N_E \)` indicating that there are multiple experiments. When multiple experiments are present, Dakota will expand the number of residuals for the repeat measurement data and difference with the data accordingly. For example, if the user has five experiments in the example above with three calibration terms, the `calibration_data_file` would need to contain five rows (one for each experiment), and each row should contain three experimental data values that
will be differenced with respect to the appropriate model response. In this example, \( N_E = 5 \). To summarize, Dakota will calculate the sum of the squared residuals as:

\[
f = \sum_{i=1}^{N_E} R_i^2
\]

where the residuals now are calculated as:

\[
R_i = y_i^{\text{Model}}(\theta) - y_i^{\text{Data}}.
\]

annotated

- Keywords Area
- responses
- calibration_terms
- calibration_data_file
- annotated

Selects annotated tabular file format for experiment data

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. For experiment data files, each subsequent row contains an experiment ID, followed by data for configuration variables, observations, and/or observation errors, depending on context.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. The annotated keyword can be used to explicitly specify this.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
Examples

Import an annotated experimental data file containing a header row, leading exp_id column, and experiment data in a calibration study

```sh
responses
... scalar_data_file 'shock_experiment.dat'
      annotated
```

Example data file with two measured quantities, three experiments:

```
%exp_id   velocity  stress
1     18.23    83.21
2     34.14    93.24
3     22.41    88.92
```

custom_annotated

- Keywords Area
- responses
- calibration_terms
- calibration_data_file
- custom_annotated

Selects custom-annotated tabular file format for experiment data

Topics

This keyword is related to the topics:

- file_formats

Specification

Alias: none
Argument(s): none
Default: annotated format

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>header</td>
<td>Enable header row in custom-annotated tabular file</td>
</tr>
</tbody>
</table>
CHAPTER 6. KEYWORDS AREA

Description

A custom-annotated tabular file is a whitespace-separated text file containing experiment data, including configuration variables, observations, and/or observation errors, depending on context. For experiment import, custom-annotated allows user options for whether header row and exp_id column appear in the tabular file, thus bridging freeform and (fully) annotated.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. The custom_annotated keyword, followed by options can be used to select other formats.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import an experimental data file containing a header row, no leading exp_id column, and experiment data in a calibration study

```plaintext
responses
... scalar_data_file 'shock_experiment.dat' custom_annotated header
```

Example data file with two measured quantities, three experiments:

```
% velocity  stress  
18.23    83.21
34.14    93.24
22.41    88.92
```

```
header
```

- Keywords Area

- responses

- calibration_terms

- calibration_data_file

- custom_annotated

- header

Enable header row in custom-annotated tabular file
6.6. RESPONSES

Specification
Alias: none
Argument(s): none
Default: no header

Description
See description of parent custom_annotated

exp_id

- Keywords Area
- responses
- calibration_terms
- calibration_data_file
- custom_annotated
- exp_id

Enable experiment ID column in custom-annotated tabular file

Specification
Alias: none
Argument(s): none
Default: no exp_id column

Description
See description of parent custom_annotated

freeform

- Keywords Area
- responses
- calibration_terms
- calibration_data_file
- freeform

Selects free-form tabular file format for experiment data

Topics
This keyword is related to the topics:

- file_formats
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. For experiment data files, each row contains data for configuration variables, observations, and/or observation errors, depending on context.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. Specify freeform to instead select this format.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though freeform remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import a free-form experimental data file containing raw experiment data in a calibration study

```
responses...
scalar_data_file 'shock_experiment.dat'
  freeform
```

Example data file with two measured quantities, three experiments:

```
18.23  83.21
34.14  93.24
22.41  88.92
```

num_experiments

- Keywords Area

- responses

- calibration_terms

- calibration_data_file

- num_experiments

Add context to data: number of different experiments

Specification

Alias: none

Argument(s): INTEGER

Default: 1
6.6. RESPONSES

Description

The number of different experiments. Dakota will expand the total number of residual terms based on the number of calibration terms and the number of experiments. For example, if the number of calibration terms are five scalars, and there are three experiments, the total number of residuals in the least squares formulation will be 15. See calibration_data or calibration_data_file.

num_config_variables

- Keywords Area
- responses
- calibration_terms
- calibration_data_file
- num_config_variables

Add context to data: number of configuration variables.

Specification

Alias: none

Argument(s): INTEGER
Default: 0

Description

This is an optional description. If there are multiple experiments, there can be different configuration variables (e.g. experimental settings, boundary conditions, etc.) per experiment. See calibration_data or calibration_data_file. Note: as of Dakota 6.2, configuration variables are not used. However, the intent is to treat them as state variables which will be passed to the simulation, and not treated as parameters which are calibrated.

variance_type

- Keywords Area
- responses
- calibration_terms
- calibration_data_file
- variance_type

Add context to experiment data description by specifying the type of experimental error.

Specification

Alias: none

Argument(s): STRINGLIST
Default: none
Description

There are four options for specifying the experimental error (e.g. the measurement error in the data you provide for calibration purposes): 'none', 'scalar', 'diagonal', or 'matrix.'

If the user specifies none, Dakota will calculate a variance (sigma-squared) term. This will be a constant variance term across all of the data. If the user specifies scalar, they can provide a scalar variance per calibration term. Note that for scalar calibration terms, only 'none' or 'scalar' are options for the measurement variance. However, for field calibration terms, there are two additional options. 'diagonal' allows the user to provide a vector of measurement variances (one for each term in the calibration field). This vector corresponds to the diagonal of the full covariance matrix of measurement errors. If the user specifies 'matrix', they can provide a full covariance matrix (not just the diagonal terms), where each element(i,j) of the covariance matrix represents the covariance of the measurement error between the i-th and j-th field values.

nonlinear_inequality_constraints

- Keywords Area
- responses
- calibration_terms
- nonlinear_inequality_constraints

Group to specify nonlinear inequality constraints

Specification

Alias: num_nonlinear_inequality_constraints
Argument(s): INTEGER
Default: 0

| Required/- | Description of | Dakota Keyword | Dakota Keyword |
| Optional   | Group          |               | Description    |
| Optional   |               | lower_bounds  | Specify minimum values |
| Optional   |               | upper_bounds  | Specify maximum values |
| Optional   |               | scale_types   | Choose how each constraint is scaled |
| Optional   |               | scales        | Characteristic values for scaling |

Description

The lower_bounds and upper_bounds specifications provide the lower and upper bounds for 2-sided nonlinear inequalities of the form

\[ g_l \leq g(x) \leq g_u \]

The defaults for the inequality constraint bounds are selected so that one-sided inequalities of the form

\[ g(x) \leq 0.0 \]

result when there are no user constraint bounds specifications (this provides backwards compatibility with previous Dakota versions).
In a user bounds specification, any upper bound values greater than \( +\text{bigRealBoundSize} \) (1.e+30, as defined in Minimizer) are treated as +infinity and any lower bound values less than \( -\text{bigRealBoundSize} \) are treated as -infinity. This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since \(-\text{DBL_MAX} < -\text{bigRealBoundSize}\)). The same approach is used for nonexistent linear inequality bounds and for nonexistent design variable bounds.

The \textit{scale_types} and \textit{scales} keywords are related to scaling of \( g(x) \). See the scaling keyword in the \textit{method} section for details.

\subsection*{lower\_bounds}

- Keywords Area
- responses
- calibration\_terms
- nonlinear\_inequality\_constraints
- lower\_bounds

Specify minimum values

\subsection*{Specification}

\textbf{Alias:} nonlinear\_inequality\_lower\_bounds
\textbf{Argument(s):} REALLIST
\textbf{Default:} vector values = -infinity

\subsection*{Description}

Specify minimum values

\subsection*{upper\_bounds}

- Keywords Area
- responses
- calibration\_terms
- nonlinear\_inequality\_constraints
- upper\_bounds

Specify maximum values

\subsection*{Specification}

\textbf{Alias:} nonlinear\_inequality\_upper\_bounds
\textbf{Argument(s):} REALLIST
\textbf{Default:} vector values = 0 .
**Description**

Specify maximum values

**scale_types**

- Keywords Area
- responses
- calibration_terms
- nonlinear_inequality_constraints
- scale_types

Choose how each constraint is scaled

**Specification**

*Alias:* nonlinear_inequality_scale_types  
*Argument(s):* STRINGLIST  
*Default:* no scaling

**Description**

See the `scaling` keyword in the `method` section for details on how to use this keyword.

**scales**

- Keywords Area
- responses
- calibration_terms
- nonlinear_inequality_constraints
- scales

Characteristic values for scaling

**Specification**

*Alias:* nonlinear_inequality_scales  
*Argument(s):* REALLIST  
*Default:* 1.0 (no scaling)

**Description**

See the `scaling` keyword in the `method` section for details on how to use this keyword.
nonlinear_equality_constraints

- Keywords Area
- responses
- calibration_terms
- nonlinear_equality_constraints

Group to specify nonlinear equality constraints

Specification

Alias: num_nonlinear_equality_constraints

**Argument(s):** INTEGER

**Default:** 0

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>targets</td>
<td>Target values for the nonlinear equality constraint</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>scale_types</td>
<td>Choose how each constraint is scaled</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>scales</td>
<td>Characteristic values for scaling</td>
</tr>
</tbody>
</table>

**Description**

The targets specification provides the targets for nonlinear equalities of the form

\[ g(x) = g_t \]

and the defaults for the equality targets enforce a value of 0. for each constraint

\[ g(x) = 0.0 \]

The scale_types and scales keywords are related to scaling of \( g(x) \). See the scaling keyword in the method section for details.

targets

- Keywords Area
- responses
- calibration_terms
- nonlinear_equality_constraints
- targets

Target values for the nonlinear equality constraint
CHAPTER 6. KEYWORDS AREA

Specification

Alias: nonlinear_equality_targets

Argument(s): REALLIST
Default: vector values = 0 .

Description

The targets specification provides the targets for nonlinear equalities of the form

$$g(x) = g_t$$

and the defaults for the equality targets enforce a value of 0.0 for each constraint:

$$g(x) = 0.0$$

scale_types

- Keywords Area
- responses
- calibration_terms
- nonlinear_equality_constraints
- scale_types

Choose how each constraint is scaled

Specification

Alias: nonlinear_equality_scale_types

Argument(s): STRINGLIST
Default: no scaling

Description

See the scaling keyword in the method section for details on how to use this keyword.

scales

- Keywords Area
- responses
- calibration_terms
- nonlinear_equality_constraints
- scales

Characteristic values for scaling
6.6. RESPONSES

Specification

Alias: nonlinear_equality_scales
   Argument(s): REALLIST
   Default: 1.0 (no scaling)

Description

See the scaling keyword in the method section for details on how to use this keyword.

6.6.5 response_functions

- Keywords Area
- responses
- response_functions

Generic response type

Specification

Alias: num_response_functions
   Argument(s): INTEGER

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>scalar_responses</td>
<td>Number of scalar response functions</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>field_responses</td>
<td>Number of field responses functions</td>
</tr>
</tbody>
</table>

Description

A generic response data set is specified using response_functions. Each of these functions is simply a response quantity of interest with no special interpretation taken by the method in use.

Whereas objective, constraint, and residual functions have special meanings for optimization and least squares algorithms, the generic response function data set need not have a specific interpretation and the user is free to define whatever functional form is convenient.

Theory

This type of data set is used by uncertainty quantification methods, in which the effect of parameter uncertainty on response functions is quantified, and can also be used in parameter study and design of experiments methods (although these methods are not restricted to this data set), in which the effect of parameter variations on response functions is evaluated.
CHAPTER 6. KEYWORDS AREA

See Also

These keywords may also be of interest:

- objective_functions
- calibration_terms

scalar_responses

- Keywords Area
- responses
- response_functions
- scalar_responses

Number of scalar response functions

Specification

Alias: num_scalar_responses

Argument(s): INTEGER

Description

This keyword describes the number of scalar response functions. It is meant to be used in conjunction with field_responses, which describes the number of field response functions. The total number of response functions, both scalar and field, is given by response_functions. If only scalar responses functions are specified, it is not necessary to specify the number of scalar terms explicitly: one can simply say response_functions = 5 and get 5 scalar responses. However, if there are three scalar responses and 2 field responses, then response_functions = 5 but scalar_responses = 3 and field_responses = 2.

This type of data set is used by uncertainty quantification methods, in which the effect of parameter uncertainty on response functions is quantified, and can also be used in parameter study and design of experiments methods (although these methods are not restricted to this data set), in which the effect of parameter variations on response functions is evaluated.

See Also

These keywords may also be of interest:

- field_responses

fieldResponses

- Keywords Area
- responses
- response_functions
- field_responses

Number of field responses functions
6.6. RESPONSES

**Specification**

**Alias:** num_field_responses

**Argument(s):** INTEGER

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td></td>
<td>lengths</td>
<td>Lengths of field responses</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>num_coordinates_per_field</td>
<td>Number of independent coordinates for field responses</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>read_field_coordinates</td>
<td>Add context to data: flag to indicate that field coordinates should be read</td>
</tr>
</tbody>
</table>

**Description**

This keyword describes the number of field response functions. A field function is a set of related response values collected over a range of independent coordinate values which may or may not be specified by the user. For example, voltage over time would be a field function, where voltage is the field objective and time is the independent coordinate. Similarly, temperature over time and space would be a field response, where the independent coordinates would be both time and spatial coordinates such as (x,y) or (x,y,z), depending on the application. The main difference between scalar responses and field responses is that for field data, we plan to implement methods that take advantage of the correlation or relationship between the field values.

Note that if there is one field_response, and it has length 100 (meaning 100 values), then the user’s simulation code must return 100 values. Also, if there are both scalar and field responses, the user should specify the number of scalar responses as scalar_responses. If there are only field responses, it still is necessary to specify both response_functions = NN and field_responses = NN, where NN is the number of field responses.

This type of data set is used by uncertainty quantification methods, in which the effect of parameter uncertainty on response functions is quantified, and can also be used in parameter study and design of experiments methods (although these methods are not restricted to this data set), in which the effect of parameter variations on response functions is evaluated. Currently, field response functions will be translated back to scalar responses. So, a field of length 100 will be treated as 100 separate scalar responses. However, in future versions of Dakota, we plan to implement methods which can exploit the nature of field data.

**See Also**

These keywords may also be of interest:

- scalar_responses
- lengths
  - Keywords Area
  - responses
  - response_functions
• field_responses
• lengths

Lengths of field responses

**Specification**
Alias: none
Argument(s): INTEGERLIST

**Description**
This keyword describes the lengths of each field response. It is an integer vector of length field_responses. For example, if the field_responses = 2, an example would be lengths = 50 200, indicating that the first field response has 50 field elements but the second one has 200. The coordinate values (e.g. the independent variables) corresponding to these field responses are read in files labeled response_descriptor.coords.

**See Also**
These keywords may also be of interest:
• field_responses

**num_coordinates_per_field**
• Keywords Area
• responses
• response_functions
• field_responses
• num_coordinates_per_field

Number of independent coordinates for field responses

**Specification**
Alias: none
Argument(s): INTEGERLIST

**Description**
This keyword describes the number of independent coordinates for each field response. It is an integer vector of length field_responses. For example, if the field_responses = 2, an example would be num_coordinates_per_field = 2 1 means that the first field response has two sets of independent coordinates (perhaps x, y locations), but the second response only has one (for example, time where the field response is only dependent upon time). The actual coordinate values (e.g. the independent variables) corresponding to these field responses are defined in a file call response_descriptor.coords, where response_descriptor is the name of the individual field.
6.6. RESPONSES

See Also
These keywords may also be of interest:

- field_responses

read_field_coordinates

- Keywords Area
- responses
- response_functions
- field_responses
- read_field_coordinates

Add context to data: flag to indicate that field coordinates should be read

Specification
Alias: none
Argument(s): none

Description
Field coordinates specify independent variables (e.g., spatial or temporal coordinates) upon which the field depends. For example, the voltage level above might be a function of time, so time is the field coordinate. If the user has field coordinates to read, they need to specify read_field_coordinates. The field coordinates will then be read from a file named response_descriptor.coords, where response_descriptor is the user-provided descriptor for the field response. The number of columns in the coords file should be equal to the number of field coordinates.

6.6.6 no_gradients

- Keywords Area
- responses
- no_gradients

Gradients will not be used

Specification
Alias: none
Argument(s): none

Description
The no_gradients specification means that gradient information is not needed in the study. Therefore, it will neither be retrieved from the simulation nor computed with finite differences. The no_gradients keyword is a complete specification for this case.
**See Also**

These keywords may also be of interest:

- numerical_gradients
- analytic_gradients
- mixed_gradients

### 6.6.7 analytic_gradients

- Keywords Area
- responses
- analytic_gradients

Analysis driver will return gradients

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

The `analytic_gradients` specification means that gradient information is available directly from the simulation (finite differencing is not required). The simulation must return the gradient data in the Dakota format (enclosed in single brackets; see Dakota File Data Formats in the Users Manual[4]) for the case of file transfer of data. The `analytic_gradients` keyword is a complete specification for this case.

**See Also**

These keywords may also be of interest:

- numerical_gradients
- no_gradients
- mixed_gradients

### 6.6.8 mixed_gradients

- Keywords Area
- responses
- mixed_gradients

Gradients are needed and will be obtained from a mix of numerical and analytic sources

**Specification**

**Alias:** none

**Argument(s):** none
### Description

The `mixed_gradients` specification means that some gradient information is available directly from the simulation (analytic) whereas the rest will have to be finite differenced (numerical). This specification allows the user to make use of as much analytic gradient information as is available and then finite difference for the rest.

The `method_source`, `interval_type`, and `fd_gradient_step_size` specifications pertain to those functions listed by the `id_numerical_gradients` list.

### Examples

For example, the objective function may be a simple analytic function of the design variables (e.g., weight) whereas the constraints are nonlinear implicit functions of complex analyses (e.g., maximum stress).

### See Also

These keywords may also be of interest:

- `numerical_gradients`
- `no_gradients`
• analytic_gradients

id_numerical_gradients

• Keywords Area
• responses
• mixed_gradients
• id_numerical_gradients

Identify which numerical gradient corresponds to which response

Topics
This keyword is related to the topics:
• objective_function_pointer

Specification
Alias: none
Argument(s): INTEGERLIST

Description
The id_analytic_gradients list specifies by number the functions which have analytic gradients, and the id_numerical_gradients list specifies by number the functions which must use numerical gradients. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the id_analytic_gradients and id_numerical_gradients lists.

See Also
These keywords may also be of interest:
• id_analytic_gradients

id_analytic_gradients

• Keywords Area
• responses
• mixed_gradients
• id_analytic_gradients

Identify which analytical gradient corresponds to which response

Topics
This keyword is related to the topics:
• objective_function_pointer
6.6. RESPONSES

Specification
Alias: none
Argument(s): INTEGERLIST

Description
The id_analytic_gradients list specifies by number the functions which have analytic gradients, and the id_numerical_gradients list specifies by number the functions which must use numerical gradients. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the id_analytic_gradients and id_numerical_gradients lists.

See Also
These keywords may also be of interest:
- id_numerical_gradients

method_source
- Keywords Area
- responses
- mixed_gradients
- method_source

Specify which finite difference routine is used

Specification
Alias: none
Argument(s): none
Default: dakota

Description
The method_source setting specifies the source of the finite differencing routine that will be used to compute the numerical gradients:
- dakota (default)
- vendor

dakota denotes Dakota’s internal finite differencing algorithm and vendor denotes the finite differencing algorithm supplied by the iterator package in use (DOT, CONMIN, NPSOL, NL2SOL, NLSSOL, and OPT++ each have their own internal finite differencing routines). The dakota routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [4]).

However, the vendor setting can be desirable in some cases since certain libraries will modify their algorithm when the finite differencing is performed internally. Since the selection of the dakota routine hides the use of finite differencing from the optimizers (the optimizers are configured to accept user-supplied gradients, which...
some algorithms assume to be of analytic accuracy), the potential exists for the vendor setting to trigger the use of an algorithm more optimized for the higher expense and/or lower accuracy of finite-differencing. For example, NPSOL uses gradients in its line search when in user-supplied gradient mode (since it assumes they are inexpensive), but uses a value-based line search procedure when internally finite differencing. The use of a value-based line search will often reduce total expense in serial operations. However, in parallel operations, the use of gradients in the NPSOL line search (user-supplied gradient mode) provides excellent load balancing without need to resort to speculative optimization approaches.

In summary, then, the dakota routine is preferred for parallel optimization, and the vendor routine may be preferred for serial optimization in special cases.

### Specification

**Alias:** none  
**Argument(s):** none  
**Default:** relative

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>ignore_bounds</td>
<td>Do not respect bounds when computing gradients or Hessians.</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>Group 1</td>
<td>relative</td>
<td>Scale step size by the parameter value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>absolute</td>
<td>Do not scale step-size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bounds</td>
<td>Scale step-size by the domain of the parameter.</td>
</tr>
</tbody>
</table>

**Description**

The dakota routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [4]).

When the method_source is dakota, the user may also specify the type of scaling desired when determining the finite difference step size. The choices are absolute, bounds, and relative. For absolute, the step size will be applied as is. For bounds, it will be scaled by the range of each parameter. For relative, it will be scaled by the parameter value.
ignore_bounds

- Keywords Area
- responses
- mixed_gradients
- dakota
- ignore_bounds

Do not respect bounds when computing gradients or Hessians

**Specification**

**Alias:** none

- **Argument(s):** none
- **Default:** bounds respected

**Description**

When Dakota computes gradients or Hessians by finite differences and the variables in question have bounds, it by default chooses finite-differencing steps that keep the variables within their specified bounds. Older versions of Dakota generally ignored bounds when computing finite differences. To restore the older behavior, one can add keyword `ignore_bounds` to the response specification when `method_source dakota` (or just `dakota`) is also specified.

In forward difference or backward difference computations, honoring bounds is straightforward. To honor bounds when approximating $\frac{\partial f}{\partial x_i}$, i.e., component $i$ of the gradient of $f$, by central differences, Dakota chooses two steps $h_1$ and $h_2$ with $h_1 \neq h_2$, such that $x + h_1 e_i$ and $x + h_2 e_i$ both satisfy the bounds, and then computes

$$\frac{\partial f}{\partial x_i} \approx \frac{h_2^2(f_1 - f_0) - h_1^2(f_2 - f_0)}{h_1 h_2 (h_2 - h_1)},$$

with $f_0 = f(x)$, $f_1 = f(x + h_1 e_i)$, and $f_2 = f(x + h_2 e_i)$.

**relative**

- Keywords Area
- responses
- mixed_gradients
- dakota
- relative

Scale step size by the parameter value

**Specification**

**Alias:** none

- **Argument(s):** none
**Description**

Scale step size by the parameter value

**absolute**

- Keywords Area
- responses
- mixed_gradients
- dakota
- absolute

Do not scale step-size

**Specification**

*Alias*: none  
*Argument(s)*: none  
*Default*: relative

**Description**

Do not scale step-size

**bounds**

- Keywords Area
- responses
- mixed_gradients
- dakota
- bounds

Scale step-size by the domain of the parameter

**Specification**

*Alias*: none  
*Argument(s)*: none

**Description**

Scale step-size by the domain of the parameter
6.6. RESPONSES

vendor

- Keywords Area
- responses
- mixed_gradients
- vendor

Use non-Dakota fd algorithm

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

interval_type

- Keywords Area
- responses
- mixed_gradients
- interval_type

Specify how to compute gradients and hessians

Specification

Alias: none

Argument(s): none

Default: forward

Description

The interval_type setting is used to select between forward and central differences in the numerical gradient calculations. The dakota, DOT vendor, and OPT++ vendor routines have both forward and central differences available, the CONMIN and NL2SOL vendor routines support forward differences only, and the NP-SOL and NLSSOL vendor routines start with forward differences and automatically switch to central differences as the iteration progresses (the user has no control over this). The following forward difference expression

$$\nabla f(x) \approx \frac{f(x + h e_i) - f(x)}{h}$$

and the following central difference expression

$$\nabla f(x) \approx \frac{f(x + h e_i) - f(x - h e_i)}{2h}$$

are used to estimate the $i^{th}$ component of the gradient vector.
forward
- Keywords Area
- responses
- mixed_gradients
- forward

Use forward differences

**Specification**

**Alias**: none
**Argument(s)**: none
**Default**: forward

**Description**

See parent page for usage notes.

central
- Keywords Area
- responses
- mixed_gradients
- central

Use central differences

**Specification**

**Alias**: none
**Argument(s)**: none

**Description**

See parent page for usage notes.

**fd_step_size**
- Keywords Area
- responses
- mixed_gradients
- fd_step_size

Step size used when computing gradients and Hessians
6.6. RESPONSES

Specification

Alias: fd_gradient_step_size
Argument(s): REALIST
Default: 0.001

Description

fd_gradient_step_size specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of absolute, the differencing interval will be fd_gradient_step_size.

For Dakota with an interval scaling type of bounds, the differencing intervals are computed by multiplying fd_gradient_step_size with the range of the parameter. For Dakota (with an interval scaling type of relative), DOT, CONMIN, and OPT++, the differencing intervals are computed by multiplying the fd_gradient_step_size with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ all use .01*fd_gradient_step_size as their minimum absolute differencing interval. With a fd_gradient_step_size = .001, for example, Dakota, DOT, CONMIN, and OPT++ will use intervals of .001*current parameter value with a minimum interval of 1.e-5. NPSOL and NLSSOL use a different formula for their finite difference intervals: fd_gradient_step_size*(1+|current parameter value|). This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.

6.6.9 numerical_gradients

- Keywords Area
- responses
- numerical_gradients

Gradients are needed and will be approximated by finite differences

Specification

Alias: none
Argument(s): none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>method_source</td>
<td>Specify which finite difference routine is used</td>
</tr>
</tbody>
</table>
### Description

The `numerical_gradients` specification means that gradient information is needed and will be computed with finite differences using either the native or one of the vendor finite differencing routines.

### See Also

These keywords may also be of interest:

- `no_gradients`
- `analytic_gradients`
- `mixed_gradients`

### method_source

- `Keywords Area`
- `responses`
- `numerical_gradients`
- `method_source`

Specify which finite difference routine is used

### Specification

**Alias:** none

**Argument(s):** none

**Default:** dakota
6.6. RESPONSES

Description

The `method_source` setting specifies the source of the finite differencing routine that will be used to compute the numerical gradients:

- **dakota** (default)
- **vendor**

`dakota` denotes Dakota’s internal finite differencing algorithm and `vendor` denotes the finite differencing algorithm supplied by the iterator package in use (DOT, CONMIN, NPSOL, NL2SOL, NLSSOL, and OPT++ each have their own internal finite differencing routines). The `dakota` routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [4]).

However, the `vendor` setting can be desirable in some cases since certain libraries will modify their algorithm when the finite differencing is performed internally. Since the selection of the `dakota` routine hides the use of finite differencing from the optimizers (the optimizers are configured to accept user-supplied gradients, which some algorithms assume to be of analytic accuracy), the potential exists for the `vendor` setting to trigger the use of an algorithm more optimized for the higher expense and/or lower accuracy of finite-differencing. For example, NPSOL uses gradients in its line search when in user-supplied gradient mode (since it assumes they are inexpensive), but uses a value-based line search procedure when internally finite differencing. The use of a value-based line search will often reduce total expense in serial operations. However, in parallel operations, the use of gradients in the NPSOL line search (user-supplied gradient mode) provides excellent load balancing without need to resort to speculative optimization approaches.

In summary, then, the `dakota` routine is preferred for parallel optimization, and the `vendor` routine may be preferred for serial optimization in special cases.

dakota

- **Keywords Area**
- **responses**
- **numerical_gradients**
- **dakota**

(Default) Use internal Dakota finite differences algorithm

Specification

**Alias:** none

**Argument(s):** none

**Default:** relative
### Description

The **dakota** routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [4]).

When the `method_source` is `dakota`, the user may also specify the type of scaling desired when determining the finite difference step size. The choices are `absolute`, `bounds`, and `relative`. For `absolute`, the step size will be applied as is. For `bounds`, it will be scaled by the range of each parameter. For `relative`, it will be scaled by the parameter value.

**ignore_bounds**
- **Keywords Area**
- **responses**
- **numerical_gradients**
- **dakota**
- **ignore_bounds**

Do not respect bounds when computing gradients or Hessians

### Specification

Alias: none  
Argument(s): none  
Default: bounds respected

### Description

When Dakota computes gradients or Hessians by finite differences and the variables in question have bounds, it by default chooses finite-differencing steps that keep the variables within their specified bounds. Older versions of Dakota generally ignored bounds when computing finite differences. To restore the older behavior, one can add keyword **ignore_bounds** to the response specification when `method_source dakota` (or just `dakota`) is also specified.
In forward difference or backward difference computations, honoring bounds is straightforward.

To honor bounds when approximating $\frac{\partial f}{\partial x_i}$, i.e., component $i$ of the gradient of $f$, by central differences, Dakota chooses two steps $h_1$ and $h_2$ with $h_1 \neq h_2$, such that $x + h_1 e_i$ and $x + h_2 e_i$ both satisfy the bounds, and then computes

$$\frac{\partial f}{\partial x_i} \approx \frac{h_2^2 (f_1 - f_0) - h_1^2 (f_2 - f_0)}{h_1 h_2 (h_2 - h_1)},$$

with $f_0 = f(x)$, $f_1 = f(x + h_1 e_i)$, and $f_2 = f(x + h_2 e_i)$.

**relative**

- Keywords Area
- responses
- numerical_gradients
- dakota
- relative

Scale step size by the parameter value

**Specification**

**Alias:** none

**Argument(s):** none

**Description**

Scale step size by the parameter value

**absolute**

- Keywords Area
- responses
- numerical_gradients
- dakota
- absolute

Do not scale step-size

**Specification**

**Alias:** none

**Argument(s):** none

**Default:** relative

**Description**

Do not scale step-size
bounds
  • Keywords Area
  • responses
  • numerical_gradients
  • dakota
  • bounds

Scale step-size by the domain of the parameter

Specification
Alias: none
Argument(s): none

Description
Scale step-size by the domain of the parameter

vendor
  • Keywords Area
  • responses
  • numerical_gradients
  • vendor

Use non-Dakota fd algorithm

Specification
Alias: none
Argument(s): none

Description
See parent page for usage notes.

interval_type
  • Keywords Area
  • responses
  • numerical_gradients
  • interval_type

Specify how to compute gradients and hessians
6.6. RESPONSES

Specification

Alias: none
   Argument(s): none
   Default: forward

Description

The interval_type setting is used to select between forward and central differences in the numerical gradient calculations. The dakota, DOT vendor, and OPT++ vendor routines have both forward and central differences available, the CONMIN and NL2SOL vendor routines support forward differences only, and the NP-SOL and NLSOL vendor routines start with forward differences and automatically switch to central differences as the iteration progresses (the user has no control over this). The following forward difference expression

$$\nabla f(x) \approx \frac{f(x + he_i) - f(x)}{h}$$

and the following central difference expression

$$\nabla f(x) \approx \frac{f(x + he_i) - f(x - he_i)}{2h}$$

are used to estimate the $i^{th}$ component of the gradient vector.

forward

- Keywords Area
- responses
- numerical_gradients
- forward
  Use forward differences

Specification

Alias: none
   Argument(s): none
   Default: forward

Description

See parent page for usage notes.

central

- Keywords Area
- responses
- numerical_gradients
- central
  Use central differences
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

fd_step_size

- Keywords Area
- responses
- numerical_gradients
- fd_step_size

Step size used when computing gradients and Hessians

Specification

Alias: fd_gradient_step_size

Argument(s): REALLIST

Default: 0.001

Description

fd_gradient_step_size specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of absolute, the differencing interval will be fd_gradient_step_size.

For Dakota with an interval scaling type of bounds, the differencing intervals are computed by multiplying fd_gradient_step_size with the range of the parameter. For Dakota (with an interval scaling type of relative), DOT, CONMIN, and OPT++, the differencing intervals are computed by multiplying the fd_gradient_step_size with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ will use intervals of .01*current value with a minimum interval of 1.e-5. NPSOL and NLSSOL use a different formula for their finite difference intervals: fd_gradient_step_size*(1+|current parameter value|). This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.
6.6.10 no_hessians

- Keywords Area
- responses
- no_hessians

Hessians will not be used

**Specification**

*Alias:* none

*Argument(s):* none

**Description**

The no_hessians specification means that the method does not require Dakota to manage the computation of any Hessian information. Therefore, it will neither be retrieved from the simulation nor computed by Dakota. The no_hessians keyword is a complete specification for this case. Note that, in some cases, Hessian information may still be being approximated internal to an algorithm (e.g., within a quasi-Newton optimizer such as optpp_qnewton); however, Dakota has no direct involvement in this process and the responses specification need not include it.

**See Also**

These keywords may also be of interest:

- numerical_hessians
- quasi_hessians
- analytic_hessians
- mixed_hessians

6.6.11 numerical_hessians

- Keywords Area
- responses
- numerical_hessians

Hessians are needed and will be approximated by finite differences

**Specification**

*Alias:* none

*Argument(s):* none
CHAPTER 6. KEYWORDS AREA

Required/-Optional | Description of Group | Dakota Keyword | Dakota Keyword Description
--- | --- | --- | ---
**Optional** |  | **fd_step_size** | Step size used when computing gradients and Hessians
**Optional** |  | **relative** | Scale step size by the parameter value

**Optional** (Choose One) | **Group 1** |  |
--- | --- | ---
**absolute** |  | Do not scale step-size
**bounds** |  | Scale step-size by the domain of the parameter

**Optional** (Choose One) | **difference interval (Group 2)** | **forward** | Use forward differences
--- | --- | --- | ---
**central** |  | Use central differences

**Description**

The **numerical_hessians** specification means that Hessian information is needed and will be computed with finite differences using either first-order gradient differencing (for the cases of **analytic_gradients** or for the functions identified by **id_analytic_gradients** in the case of **mixed_gradients**) or first- or second-order function value differencing (all other gradient specifications). In the former case, the following expression

\[ \nabla^2 f(x)_i \approx \frac{\nabla f(x + he_i) - \nabla f(x)}{h} \]

estimates the \( i^{th} \) Hessian column, and in the latter case, the following expressions

\[ \nabla^2 f(x)_{i,j} \approx \frac{f(x + he_i + he_j) - f(x + he_i) - f(x - he_j) + f(x)}{h_i h_j} \]

and

\[ \nabla^2 f(x)_{i,j} \approx \frac{f(x + he_i + he_j) - f(x + he_i - he_j) - f(x - he_i + he_j) - f(x - he_i - he_j)}{4h^2} \]

provide first- and second-order estimates of the \( ij^{th} \) Hessian term. Prior to Dakota 5.0, Dakota always used second-order estimates. In Dakota 5.0 and newer, the default is to use first-order estimates (which honor bounds on the variables and require only about a quarter as many function evaluations as do the second-order estimates), but specifying **central** after **numerical_hessians** causes Dakota to use the old second-order estimates, which do not honor bounds. In optimization algorithms that use Hessians, there is little reason to use second-order differences in computing Hessian approximations.

**See Also**

These keywords may also be of interest:

- **no_hessians**
- **quasi_hessians**
6.6. RESPONSES

- analytic_hessians
- mixed_hessians

**fd_step_size**

- Keywords Area
- responses
- numerical_hessians
- fd_step_size

Step size used when computing gradients and Hessians

**Specification**

**Alias:** fd_hessian_step_size

**Argument(s):** REALIST

**Default:** 0.001 (forward), 0.002 (central)

**Description**

*fd_gradient_step_size* specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of *absolute*, the differencing interval will be *fd_gradient_step_size*.

For Dakota with an interval scaling type of *bounds*, the differencing intervals are computed by multiplying *fd_gradient_step_size* with the range of the parameter. For Dakota (with an interval scaling type of *relative*), DOT, CONMIN, and OPT++, the differencing intervals are computed by multiplying the *fd_gradient_step_size* with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ all use $0.01 \times \text{fd_gradient_step_size}$ as their minimum absolute differencing interval. With a *fd_gradient_step_size* = 0.001, for example, Dakota, DOT, CONMIN, and OPT++ will use intervals of 0.001 *current value* with a minimum interval of 1.e-5. NPSOL and NLSSOL use a different formula for their finite difference intervals: $\text{fd_gradient_step_size} \times (1 + |\text{current parameter value}|)$. This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.

**relative**

- Keywords Area
- responses
- numerical_hessians
- relative

Scale step size by the parameter value
CHAPTER 6. KEYWORDS AREA

Specification
Alias: none
Argument(s): none

Description
Scale step size by the parameter value

absolute
- Keywords Area
- responses
- numerical_hessians
- absolute

Do not scale step-size

Specification
Alias: none
Argument(s): none
Default: relative

Description
Do not scale step-size

bounds
- Keywords Area
- responses
- numerical_hessians
- bounds

Scale step-size by the domain of the parameter

Specification
Alias: none
Argument(s): none

Description
Scale step-size by the domain of the parameter
forward
  • Keywords Area
  • responses
  • numerical_hessians
  • forward

Use forward differences

Specification
Alias: none
  Argument(s): none
  Default: forward

Description
See parent page for usage notes.

central
  • Keywords Area
  • responses
  • numerical_hessians
  • central

Use central differences

Specification
Alias: none
  Argument(s): none

Description
See parent page for usage notes.

6.6.12 quasi_hessians
  • Keywords Area
  • responses
  • quasi_hessians

Hessians are needed and will be approximated by secant updates (BFGS or SR1) from a series of gradient evaluations
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none
  Argument(s): none
### Description

The `quasi_hessians` specification means that Hessian information is needed and will be approximated using secant updates (sometimes called "quasi-Newton updates", though any algorithm that approximates Newton’s method is a quasi-Newton method).

Compared to finite difference numerical Hessians, secant approximations do not expend additional function evaluations in estimating all of the second-order information for every point of interest. Rather, they accumulate approximate curvature information over time using the existing gradient evaluations.

The supported secant approximations include the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update (specified with the keyword `bfgs`) and the Symmetric Rank 1 (SR1) update (specified with the keyword `sr1`).

### See Also

These keywords may also be of interest:

- `no_hessians`
- `numerical_hessians`
- `analytic_hessians`
- `mixed_hessians`

### Specification

**Alias:** none

**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional Required (Choose One)</th>
<th>Description of Group 1</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td></td>
<td><code>bfgs</code></td>
<td>Use BFGS method to compute quasi-hessians</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>sr1</code></td>
<td>Use the Symmetric Rank 1 update method to compute quasi-Hessians</td>
</tr>
</tbody>
</table>
Broyden-Fletcher-Goldfarb-Shanno (BFGS) update will be used to compute quasi-Hessians.

\[ B_{k+1} = B_k - \frac{B_k s_k y_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \]

where \( B_k \) is the \( k^{th} \) approximation to the Hessian, \( s_k = x_{k+1} - x_k \) is the step and \( y_k = \nabla f_{k+1} - \nabla f_k \) is the corresponding yield in the gradients.

**Notes**

- Initial scaling of \( \frac{y_k^T y_k}{y_k^T s_k} I \) is used for \( B_0 \) prior to the first update.
- Numerical safeguarding is used to protect against numerically small denominators within the updates.
- This safeguarding skips the update if \( |y_k^T s_k| < 10^{-6} s_k^T B_k s_k \)
- Additional safeguarding can be added using the `damped` option, which utilizes an alternative damped BF-GS update when the curvature condition \( y_k^T s_k > 0 \) is nearly violated.

**damped**

- Keywords Area
- responses
- quasi_hessians
- bfgs
- damped

Numerical safeguarding for BFGS updates

**Specification**

Alias: none  
Argument(s): none  
Default: undamped BFGS

**Description**

See parent page.
6.6. RESPONSES

sr1

- Keywords Area
- responses
- quasi_hessians
- sr1

Use the Symmetric Rank 1 update method to compute quasi-Hessians

**Specification**

Alias: none

**Argument(s):** none

**Description**

The Symmetric Rank 1 (SR1) update (specified with the keyword `sr1`) will be used to compute quasi-Hessians.

\[ B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \]

where \( B_k \) is the \( k^{th} \) approximation to the Hessian, \( s_k = x_{k+1} - x_k \) is the step and \( y_k = \nabla f_{k+1} - \nabla f_k \) is the corresponding yield in the gradients.

**Notes**

- Initial scaling of \( \frac{y_k^T s_k}{s_k^T s_k} I \) is used for \( B_0 \) prior to the first update.
- Numerical safeguarding is used to protect against numerically small denominators within the updates.
- This safeguarding skips the update if \( |(y_k - B_k s_k)^T s_k| < 10^{-6} ||s_k||_2 ||y_k - B_k s_k||_2 \)

6.6.13 analytic_hessians

- Keywords Area
- responses
- analytic_hessians

Hessians are needed and are available directly from the analysis driver

**Specification**

Alias: none

**Argument(s):** none

**Description**

The `analytic_hessians` specification means that Hessian information is available directly from the simulation. The simulation must return the Hessian data in the Dakota format (enclosed in double brackets; see Dakota File Data Formats in Users Manual [4]) for the case of file transfer of data. The `analytic_hessians` keyword is a complete specification for this case.
See Also

These keywords may also be of interest:

- no_hessians
- numerical_hessians
- quasi_hessians
- mixed_hessians

### 6.6.14 mixed_hessians

- Keywords Area
- responses
- mixed_hessians

Hessians are needed and will be obtained from a mix of numerical, analytic, and "quasi" sources

#### Specification

Alias: none

Argument(s): none

<table>
<thead>
<tr>
<th>Required/-Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>id_numerical_hessians</td>
<td>Identify which numerical-Hessian corresponds to which response</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td></td>
<td>relative</td>
<td>Scale step size by the parameter value</td>
</tr>
<tr>
<td>Optional (Choose One)</td>
<td>difference interval (Group 2)</td>
<td>absolute</td>
<td>Do not scale step-size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bounds</td>
<td>Scale step-size by the domain of the parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>forward</td>
<td>Use forward differences</td>
</tr>
<tr>
<td></td>
<td></td>
<td>central</td>
<td>Use central differences</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>id_quasi_hessians</td>
<td>Identify which quasi-Hessian corresponds to which response</td>
</tr>
</tbody>
</table>
6.6. RESPONSES

### Optional

<table>
<thead>
<tr>
<th>id_analytic_hessians</th>
<th>Identify which analytical Hessian corresponds to which response</th>
</tr>
</thead>
</table>

#### Description

Hessian availability must be specified with either `no_hessians`, `numerical_hessians`, `quasi_hessians`, `analytic_hessians`, or `mixed_hessians`.

The `mixed_hessians` specification means that some Hessian information is available directly from the simulation (analytic) whereas the rest will have to be estimated by finite differences (numerical) or approximated by secant updating. As for mixed gradients, this specification allows the user to make use of as much analytic information as is available and then estimate/approximate the rest.

The `id_analytic_hessians` list specifies by number the functions which have analytic Hessians, and the `id_numerical_hessians` and `id_quasi_hessians` lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_hessians`, `id_numerical_hessians`, and `id_quasi_hessians` lists.

The `fd_hessian_step_size` and `bfgs`, `damped_bfgs`, or `sr1` secant update selections are as described previously in responses and pertain to those functions listed by the `id_numerical_hessians` and `id_quasi_hessians` lists.

#### See Also

These keywords may also be of interest:

- `no_hessians`
- `numerical_hessians`
- `quasi_hessians`
- `analytic_hessians`

#### id_numerical_hessians

- Keywords Area
  - `responses`
  - `mixed_hessians`
  - `id_numerical_hessians`

  Identify which numerical-Hessian corresponds to which response

#### Topics

This keyword is related to the topics:

- `objective_function_pointer`
Specification

Alias: none

Argument(s): INTEGERLIST
### Description

The `id_analytic_hessians` list specifies by number the functions which have analytic Hessians, and the `id_numerical_hessians` and `id_quasi_hessians` lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_hessians`, `id_numerical_hessians`, and `id_quasi_hessians` lists.

### See Also

These keywords may also be of interest:

- `id_analytic_hessians`
- `id_quasi_hessians`

### fd_step_size

- **Keywords Area**
- `responses`
- `mixed_hessians`
- `id_numerical_hessians`
- `fd_step_size`

Step size used when computing gradients and Hessians

### Specification

**Alias:** `fd.hessian_step_size`

**Argument(s):** `REALLIST`

**Default:** 0.001 (forward), 0.002 (central)

### Description

`fd.gradient_step_size` specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of `absolute`, the differencing interval will be `fd.gradient_step_size`.

For Dakota with an interval scaling type of `bounds`, the differencing intervals are computed by multiplying `fd.gradient_step_size` with the range of the parameter. For Dakota (with an interval scaling type of...
relate), DOT, CONMIN, and OPT++, the differencing intervals are computed by multiplying the \texttt{fd-gradient-step-size} with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ all use \(0.01\times\text{fd gradient step size}\) as their minimum absolute differencing interval. With a \(\text{fd.gradient.step.size} = 0.001\), for example, Dakota, DOT, CONMIN, and OPT++ will use intervals of \(0.001\times\text{current value}\) with a minimum interval of \(1.0\times10^{-5}\). NPSOL and NLSSOL use a different formula for their finite difference intervals: \(\text{fd.gradient.step.size}\times(1+|\text{current parameter value}|)\). This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.

relative

- \textbf{Keywords Area}
- \textbf{responses}
- \textbf{mixed.hessians}
- \textbf{relative}

Scale step size by the parameter value

**Specification**

Alias: none

\textbf{Argument(s):} none

**Description**

Scale step size by the parameter value

\textbf{absolute}

- \textbf{Keywords Area}
- \textbf{responses}
- \textbf{mixed.hessians}
- \textbf{absolute}

Do not scale step-size

**Specification**

Alias: none

\textbf{Argument(s):} none

\textbf{Default:} relative

**Description**

Do not scale step-size
6.6. RESPONSES

bounds

- Keywords Area
- responses
- mixed_hessians
- bounds

Scale step-size by the domain of the parameter

Specification

Alias: none
Argument(s): none

Description

Scale step-size by the domain of the parameter

forward

- Keywords Area
- responses
- mixed_hessians
- forward

Use forward differences

Specification

Alias: none
Argument(s): none
Default: forward

Description

See parent page for usage notes.

central

- Keywords Area
- responses
- mixed_hessians
- central

Use central differences
CHAPTER 6. KEYWORDS AREA

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

id_quasi_hessians

- Keywords Area
- responses
- mixed_hessians
- id_quasi_hessians

Identify which quasi-Hessian corresponds to which response

Topics

This keyword is related to the topics:

- objective_function_pointer

Specification

Alias: none

Argument(s): INTEGERLIST

<p>| Required/- | Description of | Dakota Keyword | Dakota Keyword |</p>
<table>
<thead>
<tr>
<th>Optional Required (Choose One)</th>
<th>Group 1</th>
<th>bfgs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>sr1</td>
<td>Use the Symmetric Rank 1 update method to compute quasi-Hessians</td>
</tr>
</tbody>
</table>

Description

The id_analytic_hessians list specifies by number the functions which have analytic Hessians, and the id_numerical_hessians and id_quasi_hessians lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the id_analytic_hessians, id_numerical_hessians, and id_quasi_hessians lists.
6.6. RESPONSES

See Also

These keywords may also be of interest:

- `id_numerical_hessians`
- `id_analytic_hessians`

`bfgs`

- Keywords Area
- responses
- mixed_hessians
- `id_quasi_hessians`
- `bfgs`

Use BFGS method to compute quasi-hessians

**Specification**

**Alias:** none  
**Argument(s):** none

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Description of Group</th>
<th>Dakota Keyword</th>
<th>Dakota Keyword Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td></td>
<td>damped</td>
<td>Numerical safeguarding for BFGS updates</td>
</tr>
</tbody>
</table>

**Description**

Broyden-Fletcher-Goldfarb-Shanno (BFGS) update will be used to compute quasi-Hessians.

\[
B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}
\]

where \(B_k\) is the \(k^{th}\) approximation to the Hessian, \(s_k = x_{k+1} - x_k\) is the step and \(y_k = \nabla f_{k+1} - \nabla f_k\) is the corresponding yield in the gradients.

**Notes**

- Initial scaling of \(\frac{y_k^T y_k}{y_k^T s_k} I\) is used for \(B_0\) prior to the first update.
- Numerical safeguarding is used to protect against numerically small denominators within the updates.
- This safeguarding skips the update if \(|y_k^T s_k| < 10^{-6} s_k^T B_k s_k\)
- Additional safeguarding can be added using the `damped` option, which utilizes an alternative damped BFGS update when the curvature condition \(y_k^T s_k > 0\) is nearly violated.
CHAPTER 6. KEYWORDS AREA

damped
• Keywords Area
• responses
• mixed_hessians
• id_quasi_hessians
• bfgs
• damped

Numerical safeguarding for BFGS updates

Specification
Alias: none
Argument(s): none
Default: undamped BFGS

Description
See parent page.

sr1
• Keywords Area
• responses
• mixed_hessians
• id_quasi_hessians
• sr1

Use the Symmetric Rank 1 update method to compute quasi-Hessians

Specification
Alias: none
Argument(s): none

Description
The Symmetric Rank 1 (SR1) update (specified with the keyword sr1) will be used to compute quasi-Hessians.

\[ B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \]

where \( B_k \) is the \( k^{th} \) approximation to the Hessian, \( s_k = x_{k+1} - x_k \) is the step and \( y_k = \nabla f_{k+1} - \nabla f_k \) is the corresponding yield in the gradients.

Notes
Initial scaling of $\frac{y_k^T s_k}{y_k^T s_k} I$ is used for $B_0$ prior to the first update.

Numerical safeguarding is used to protect against numerically small denominators within the updates.

This safeguarding skips the update if $|(y_k - B_k s_k)^T s_k| < 10^{-6}||s_k||_2||y_k - B_k s_k||_2$

**id_analytic_hessians**

- Keywords Area
- responses
- mixed_hessians
- id_analytic_hessians

Identify which analytical Hessian corresponds to which response

**Topics**

This keyword is related to the topics:

- objective_function_pointer

**Specification**

**Alias:** none

**Argument(s):** INTEGERLIST

**Description**

The `id_analytic_hessians` list specifies by number the functions which have analytic Hessians, and the `id_numerical_hessians` and `id_quasi_hessians` lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_hessians`, `id_numerical_hessians`, and `id_quasi_hessians` lists.

**See Also**

These keywords may also be of interest:

- `id_numerical_hessians`
- `id_quasi_hessians`
Bibliography


2259


[74] K. Schittkowski. NLPQLP: A fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search – user’s guide. Technical report, Department of Mathematics, University of Bayreuth, Bayreuth, Germany, 2004. 150


